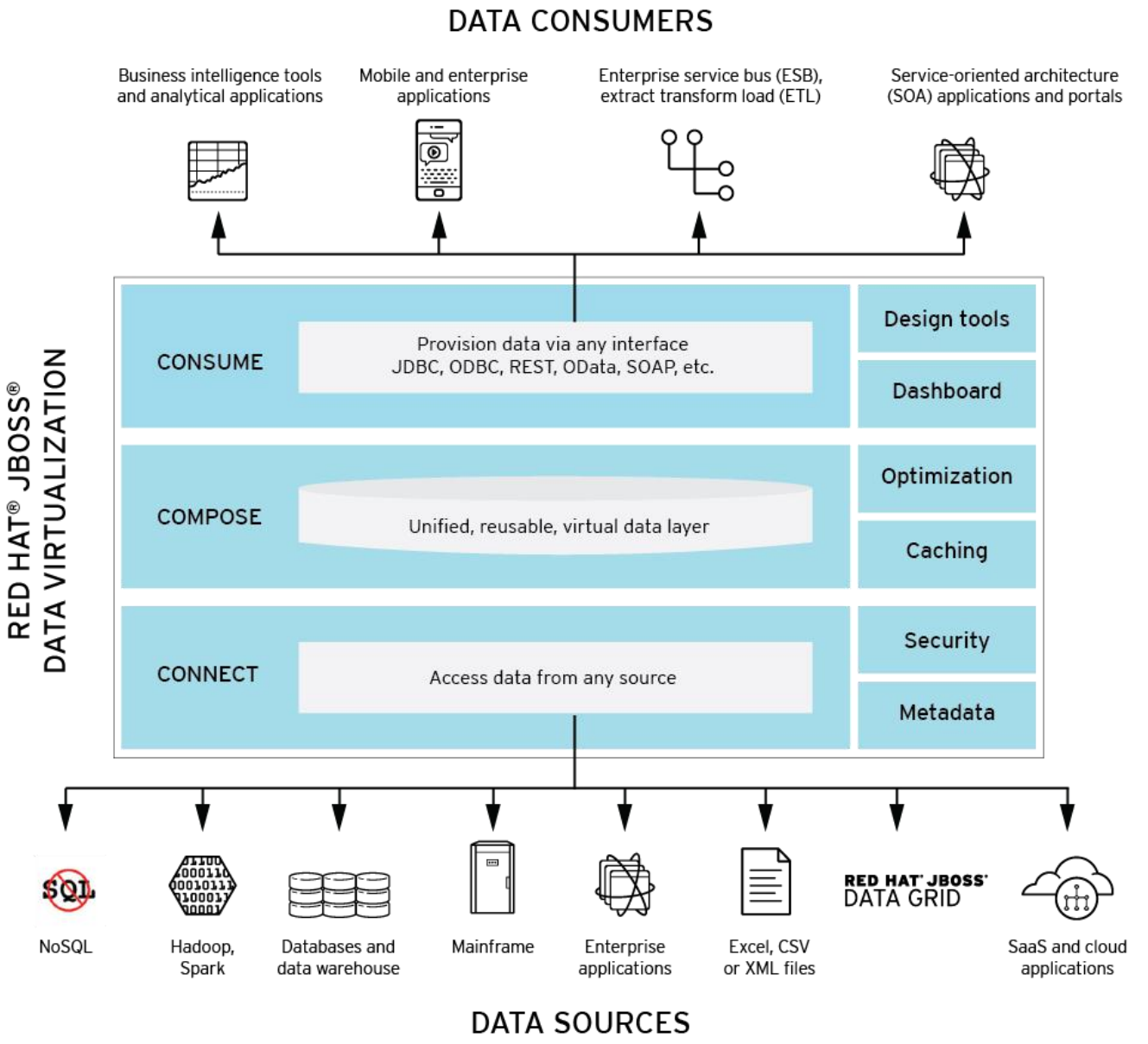
适用于数字化医院和区域医疗信息系统的多标准集成集成交换引擎

用户帮助手册

二〇一四年八月一十八日

目录


DATA CONSUMERS
Business intelligence tools and analytical applications
Mobile and enterprise applications
Enterprise service bus (ESB), extract transform load (ETL)
Service-oriented architecture (SOA) applications and portals
RED HAT® JBOSS® DATA VIRTUALIZATION
CONSUME
Provision data via any interface JDBC, ODBC, REST, OData, SOAP, etc.
Design tools
Dashboard
COMPOSE
Unified, reusable, virtual data layer
Optimization
Caching
CONNECT
Access data from any source
Security
Metadata
NoSQL
Hadoop, Spark
Databases and data warehouse
Mainframe
Enterprise applications
Excel, CSV or XML files
RED HAT JBOSS DATA GRID
SaaS and cloud applications
DATA SOURCES
JB0041-2

1 概述

“医易通”在四川省科技厅和广汉市人民医院联合支持下，由电子科技大学数字化医疗实验室在吸收国外十多年先进成果的基础上研发的一个多标准兼容医疗卫生信息系统集成交换引擎;它可以是一个松耦合的系统互联互通软总线;也可以是一个构建 SOA 架构的卫生信息平台的中间件.

使用医易通，可以快速建立一个标准化的、消除信息孤岛的医疗卫生信息集成平台和系统。方便用户和开发者集成现有的信息系统，高效、安全和可靠的实现系统的互联互通。

医易通管理员是通过 Java 实现的富应用风格的管理客户端，它可以完成医易通多标准集成交换网关的综合管理，包括通道监控仪表盘(通道监控和消息管理)、通道的维护、警告管理、事件管理、用户管理、系统设置、系统插件和扩展管理。

1.1 系统环境需求

客户端需求：

- Sun Java JRE 1.6.x 版本以上
- 512 MB 内存(推荐 1G)
- 200 MB 的可用硬盘空间(安装时要求有额外的可用空间支持数据库)

服务端：

不同的版本对 JAVA 的需要约有不同。3.x 版本要求 jdk1.7 以上。

数据库：根据自己情况选择。
存储空间：数据库：500G 以上。程序：1G 以上，SSD 硬盘最佳。
内存：8G 以上。推荐 16G。
网络接口： 两个以上千兆接口。

1.2 启动服务器

启动医易通有几种不同的方法。如果医易通运行在 Windows 系统中，可以使用服务器管理器来启动、关闭和重启医易通服务。如果没有安装管理器，可以使用标准的 Windows 服务控件来管理医易通服务。如果没有安装医易通服务，双击医易通根目录下的 start.bat 文件手动打开医易通服务。如果医易通不是运行在 Windows 系统中，可以手动打开相应的 start.sh 脚本启动医易通服务。

1.3 启动管理员

一旦医易通服务器已经启动，输入医易通管理员登录 URL：[https://]{gwserver_ip}:{httpPort}，默认的 http 端口是 10000。点击绿色按钮启动医易通管理员。如果医易通运行在 Windows 系统中，服务器管理器会有快速启动管理员的链接。这时可能会弹出一个或多个安全警告，提示医易通管理员没有数字签名。如果出现这种情况，只需点击“运行”按钮继续。

2 医易通管理员指南

2.1 登录

医易通管理员启动后，将立即显示登录对话框。

- **服务器**

输入医易通管理员登录页面的 URL：https://{服务器 IP}:{httpPort}，默认 https 端口是 10001。当启动 WEB 时，这总是填满的。

如果是 ip 地址，发现当没有在 hosts 表里加上对应的域名时，在有的版本中反应断断续续，僵死现象，新版本中已经解决这个问题。

- **用户名**

输入用户名登录。第一次安装医易通，只能使用用户名“admin”。

- **密码**

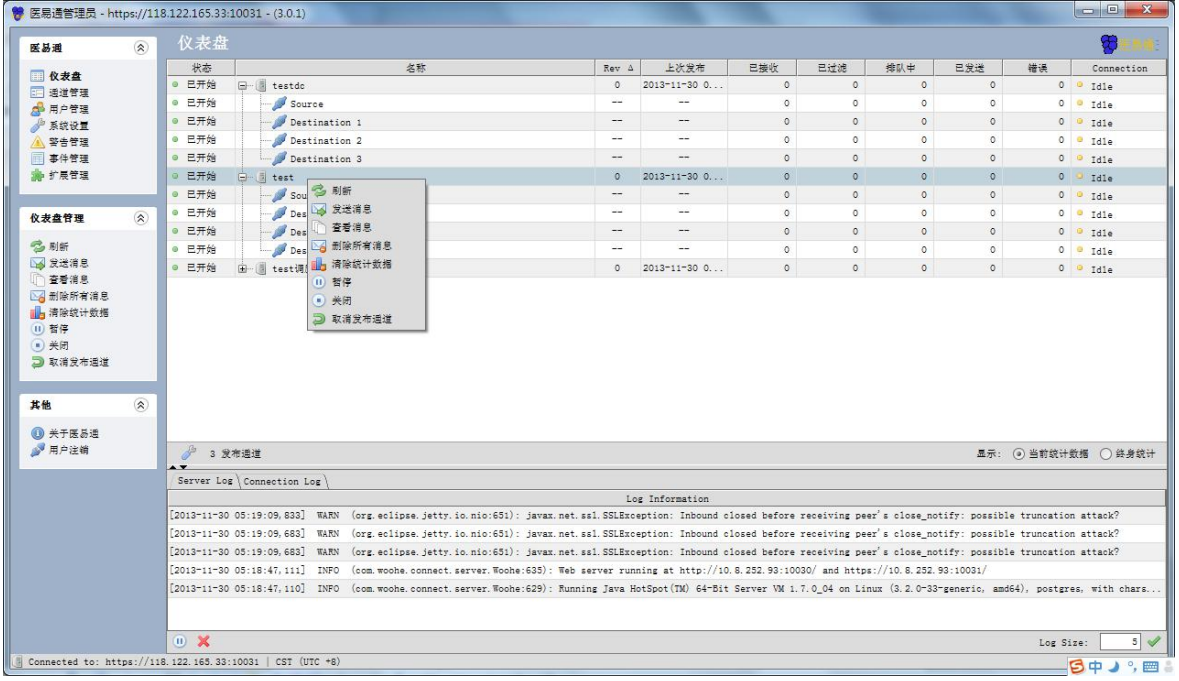
在输入用户名之后输入密码。用户名 admin 的默认密码是“admin”，建议尽快在生成机器上修改此密码。

- **登录**

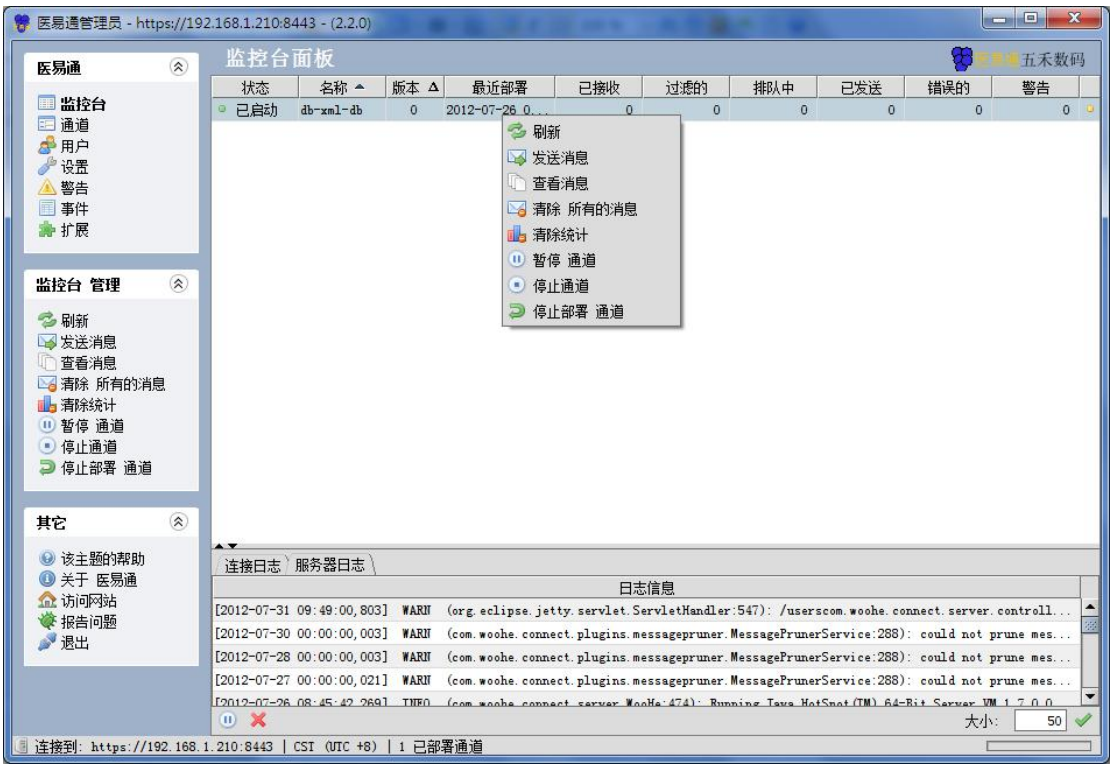
点击登录到医易通管理员。

- **退出**

点击关闭登录对话框。



2.2 仪表板



仪表板视图显示当前已部署的所有通道以及他们的连接状态，并可访问基本通道控制操作。

2.2.1 状态任务

状态任务栏可访问基本通道控制操作。

- **刷新**

更新仪表板的通道状态栏。默认情况下，此状态栏每隔 10 秒自动刷新一次。自动刷新时间可以通过设置视图进行设定。

- **启动所有通道**

  启动当前关闭的所有已部署通道。

- **关闭所有通道**

  关闭当前启动的所有已部署通道。

- **重置所有通道**

  删除所有消息并重置所有已部署通道的统计数据。

- **发送消息**

  发送消息对话框允许发送消息到所选通道，以帮助测试通道。

- **查看消息**

  显示所选通道的存储消息。参考“消息浏览器”章节的详细内容。

- **删除所有消息**

  删除所选通道的所有存储消息。

- **清除统计数据**

  清除所选通道的统计数据。

- **暂停通道**

  暂停所选通道，暂时中止其消息的处理。将接收通道的传入消息，并排入内存中。

- **关闭通道**

  关闭所选通道。此时若试图给通道发送消息将失败。

2.2.2 通道状态

通道状态栏显示所有已部署通道的状态。每一行对应于一个通道。

- **状态**

  显示此通道的状态。通道的状态有启动、关闭或暂停。

- **名称**

显示此通道的名称。

- **接收**

显示自上次清除统计数据后，此通道接收的消息数量。

- **接收**
- **过滤**

显示自上次清除统计数据后，此通道过滤（忽略）的消息数量。

- **排队**

显示此通道当前排队的消息数量。如果在消息正在排队时清除了统计数据，那么它将不能准确地代表当前排队的消息数量。

- **发送**

显示自上次清除统计数据后，此通道发送的消息数量。

- **错误**

显示自上次清除统计数据后，造成此通道上出错的消息数量。

- **连接**

显示此通道源链接器的当前状态。

2.2.3 仪表板状态面板

仪表板状态面板显示此通道最近连接状态事件的准实时视图。

- **时间标记**

显示事件的日期和时间。

- **通道**

显示发生事件的通道。如果没有选择上面的特定通道，这仅仅只是显示。

- **连接器信息**

显示此通道连接的已发生事件。

- **事件**

显示已发生的基本事件。

- **信息**

显示已发生事件的附加详细信息。

- **暂停/恢复日志**

点击这个按钮，会暂停或恢复日志显示。当暂停日志显示时，可以继续捕获日志事件。

- **清除显示的日志**

点击这个按钮，将清除日志显示。

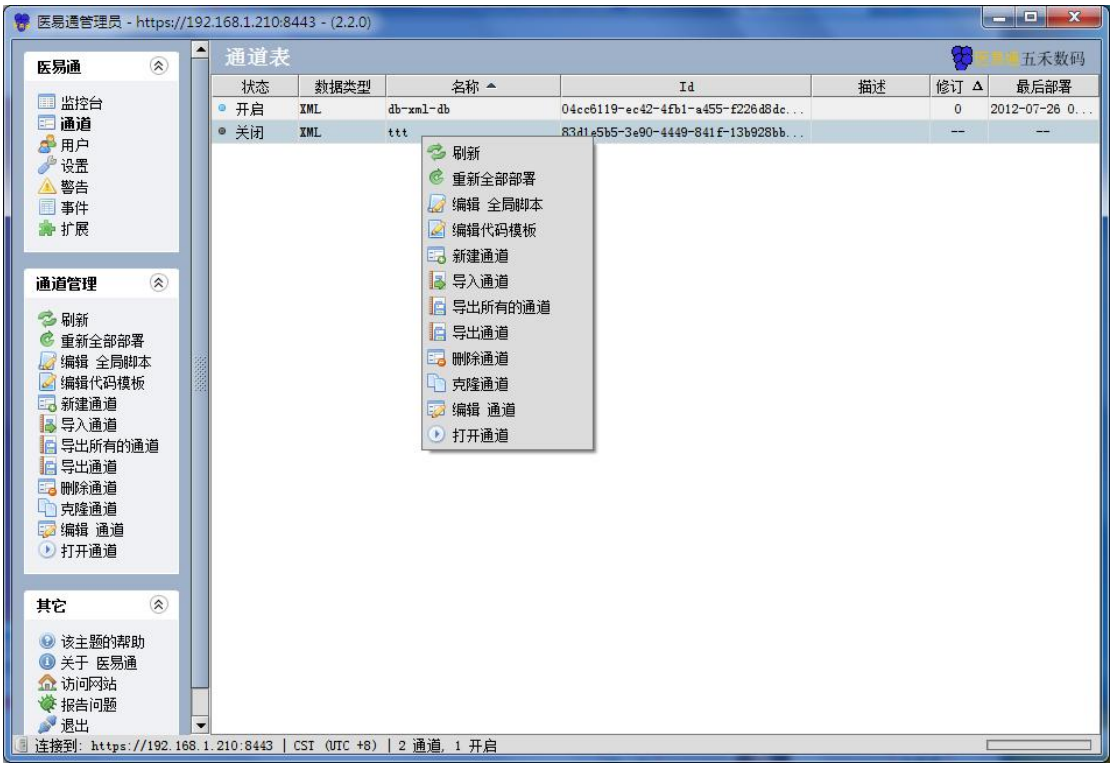
- **日志大小**

在这里输入显示事件的最大数量。

- **更改日志显示大小**

点击这个按钮，日志大小控制中的任何更改输入将生效。

2.3 通道



通道将在通道视图中创建、修改、删除和部署。

2.3.1 通道列表

通道列表显示当前定义的所有通道的摘要。

- **状态**

  显示通道是否被启用或禁用。点击“部署所有”将不部署禁用通道。

- **协议**

  显示在通道摘要标签上所选通道接收传入数据的格式。

- **名称**

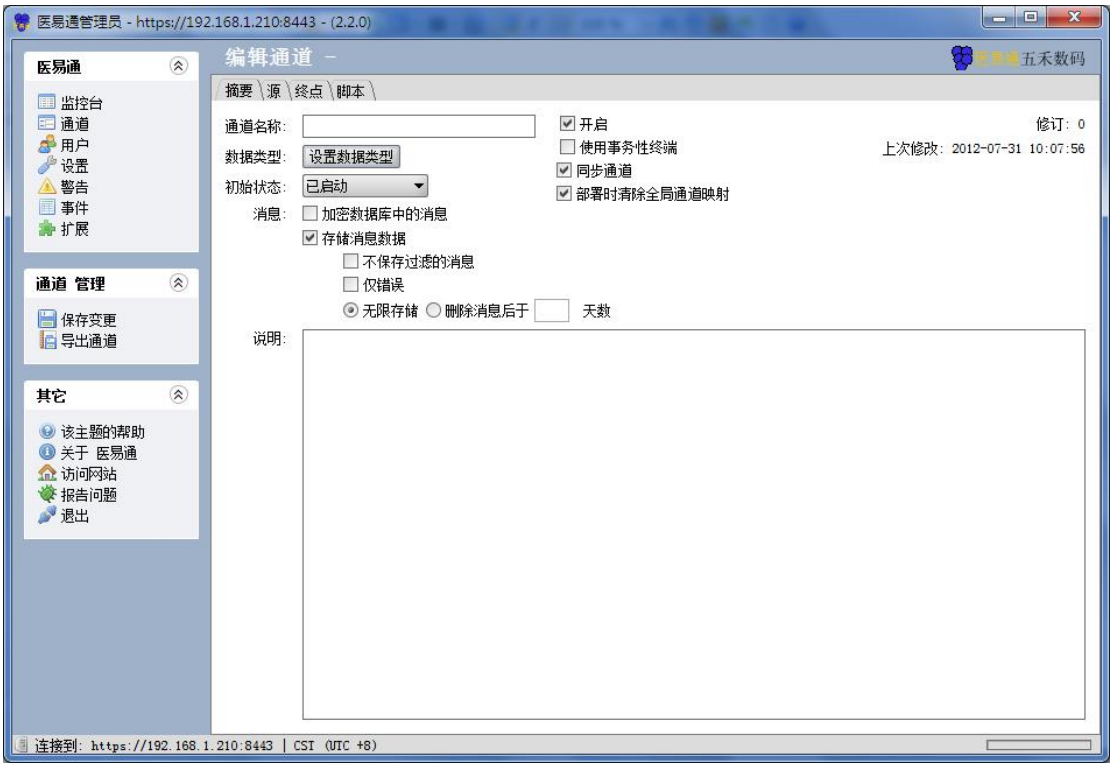
  显示在通道摘要标签上输入的通道名称。

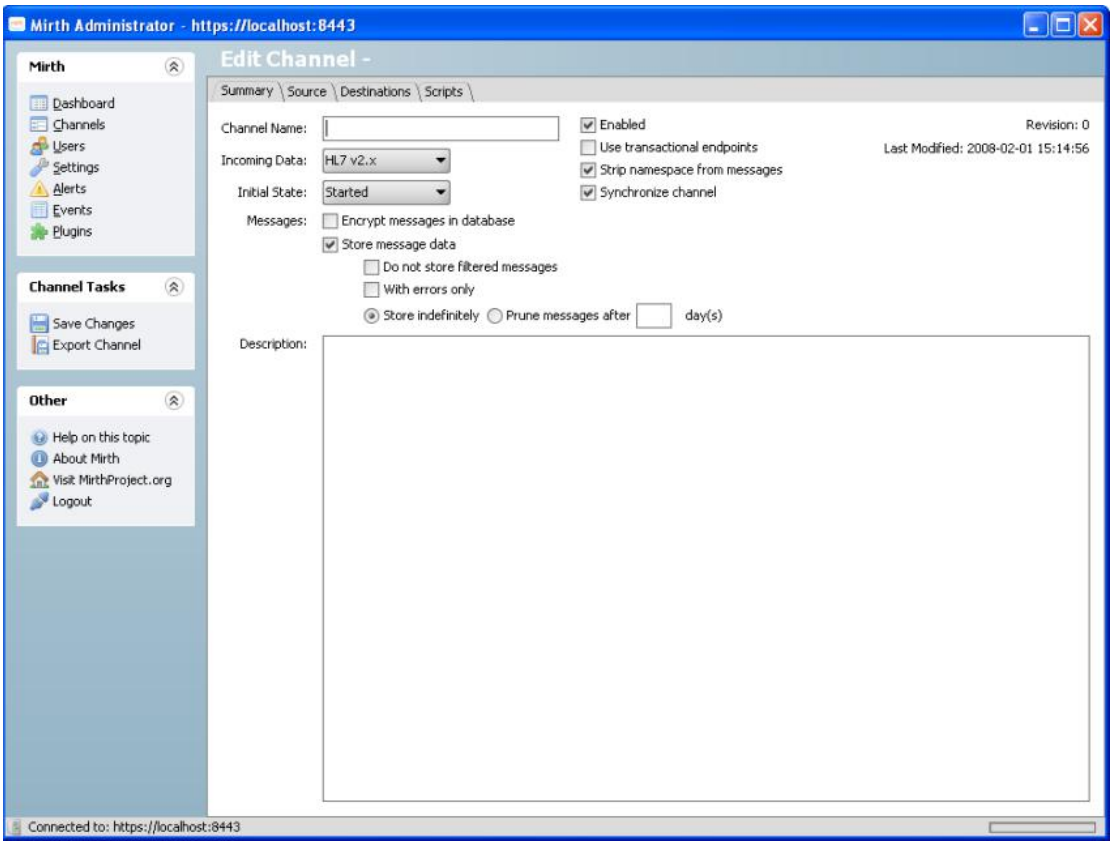
- **描述**

  显示在通道摘要标签上输入的通道描述。

2.4 编辑通道

2.4.1 摘要





- **通道任务**

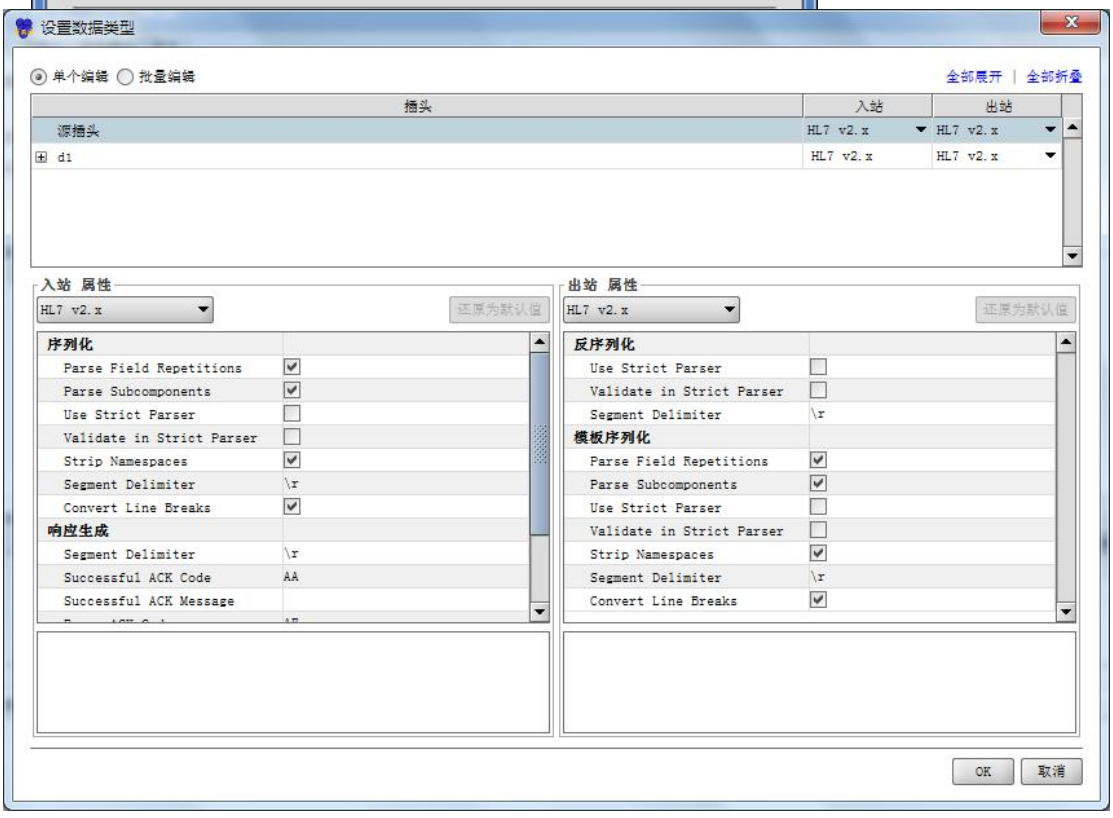
  有保存更改和导出通道两项。

- **通道名称**

分配给此通道的名称。每个通道都有一个唯一的名称，其长度不得超过 40 个字符，并且不能包含任何特殊字符。

- **传入数据**

选择通道期望传入的数据格式。



不同版本的通道数据标准类型设置界面

- **初始状态**

  选择当通道部署完后（或启动医易通时）是否立即启动通道，还是稍后手动启动。

- **启用**

  选中此复选框启用通道。不部署未启用（禁用）的通道，并且结果是无效的。

- **使用事务端点**

  选择事务端点将使所有数据库目的连接器加入到单个原子事务中。如果任何数据库的任何目的命令失败，将回滚所有其他数据库命令。使用 JavaScript，这个选项不影响数据库连接器。

- **删除消息的命名空间**

  选中此复选框自动删除传入消息的 XML 命名空间声明。对于映射通过 web 服务接收的数据，这个选项是很有用的，其中 web 服务包括多个命名空间。

- **同步通道**

  同步通道执行以下规定：

  •所有目的地将按它们在目的地列表中的定义顺序被执行。

  •源连接器将等待所有目的地完成后再发送响应(如果有的话)。

  •每个目的地的相应可以用于检查后处理程序。

  •在每个目的地处理完成后，才执行后处理程序。

  同步通道将操作利用单个执行线程接收消息。如果通道不同步，将不能保证消息的顺序。同步要求从目的地发送响应数据到源连接器。同步通道不建议适用 LLP 队列。

- **修订**

  显示通道的当前修订。第一次创建通道时修订从 0 开始，修改和保存通道时修订每次自动增加。

- **上次修改**

  显示上次更改和保存通道的时间和日期。

- **加密数据库消息**

  为保护隐私，选中此复选框将通道消息存储在加密表单中，或为了实现最高性能，不选中此复选框将消息存储在明确表单中。

- **存储消息数据**

  选中此复选框存储通道消息，或者不选中则不存储消息。不存储消息的通比存储消息的通道，每秒钟可处理更多的消息。在生成环境中运行时，这是最好的存储“只有错误”的消息。

- **不存储过滤消息**

  选中此复选框将不存储通道连接过滤器拒绝的消息，或者不选中存储该消息。

- **只有错误**

  选中此复选框存储在处理过程中有错误的消息，或者不选中存储所有消息。

- **存储无限期**

  选中此单选按钮禁止此通道修剪消息。

- **修剪消息**

  选中此单选按钮，在输入天数后删除存储的消息。也可以参阅“消息修剪”章节获取更灵活的修剪消息的方法。

- **描述**

  在这里输入此通道的任意格式的文本描述。

2.4.2 源插头

- **通道任务**

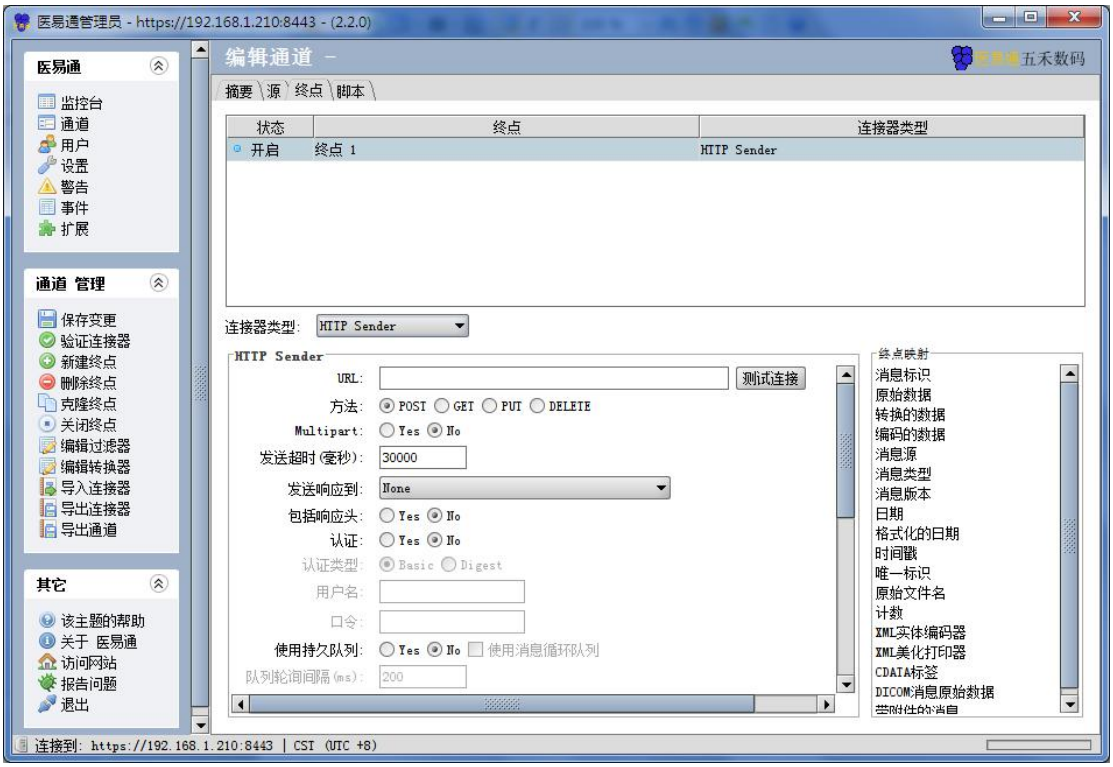
有保存更改和导出通道两项。

- **连接器类型**

  连接器类型用于选择通道读取消息的连接器类型。连接器类型是基于与目的地通信的方法。

  请参阅源连接器章节，获取源链接器及其规格的完成列表。

2.4.3 目标插头



目的地标签用来配置打开的通道如何写入出站消息。此标签包含通道的目的地连接器和其相应的过滤器和转换器。连接器设置的配置差别很大，取决于所选连接器的类型。

- **通道任务**

  有保存更改和导出通道两项。

- **目的地列表**

  显示打开通道目的地的当前列表。

- **目的地映射**

  目的地映射列表包含一个可以拖动和放到连接器设置里任何编辑控件的元素。并不是所有的元素对所有编辑控件都有意义，并不是所有的编辑控件都支持这些变量。

- **连接器类型**

  连接器类型，可以选择用于读取消息的通道的连接器类型。连接器类型取决于与目的地通信的方法。

  请参阅“目的地连接器”部分，获取源链接器及其规格的完成列表。

2.4.4 脚本

每个通道都支持与“编辑全局脚本章节”中描述的全局脚本相类似的四种脚本。在通道中使用这些脚本的原因是分散关注点。这样一来，特别是逻辑可以与一个特定的通道相关联。每个的默认版本是什么都不做。

脚本	描述
部署	每次启动机器时执行该脚本。只有全局映射对于持续数据是可用的。部署脚本可以用来构建和初始化维持医易通运行的常见对象。所有通道可以共享全局映射中的所有数据。
关闭	每次关闭机器时执行该脚本。只有全局映射对于持续数据是可用的。 关闭简本通常用来清理部署脚本创建的任何东西。
预处理	每次收到消息并在消息被规范化为 XML 之前，执行该脚本。在这里可使用通道的特定变量和全局预处理器。

后处理	发送消息后立即执行该脚本。在这里可使用通道的特定变量和全局后处理器。
	后处理器例子：
	// This script executes once after a message has been processed ?
	// $('d4')：通道 id 为 4 的通道返回响应
	
	//channelMap.put('dd7',$('d7'));
	//channelMap.put('rr7',$r);
	globalChannelMap.put('dd79',message.getConnectorMessages().get(0));
	
	//return message.toString();
	//return $('d7'); //返回标识为 7 目标插头响应
	return message.getConnectorMessages().get(7).getResponse().getContent();
	// 极而言之的例子，获取源消息对应的第 7（id 为 7）的目标插头的响应的内容，格式为 XML 字符串：
	// <response>
	// <status>SENT</status>
	// <message>testret</message> // id 为 7 的插头的 返回信息，相当于 $('d7')
	// <statusMessage>JavaScript evaluation successful.</statusMessage>
	//</response>
	//-------
	// 同等于 getResponse(), 有：getStatus（），getRawMessage(),getSent().getContent()(获取发送的消息内容）
	// getMessageId(),getEncoded(),getRaw()（原始消息）,getTransformed()(转换器转换后的）

2.5 过滤器

2.5.1 过滤器任务

- **新增规则**

  在规则列表末尾添加新规则。

- **删除规则**

  删除规则列表中当前选中的规则。

- **导入过滤器**

  用文件中存储的过滤器取代当前过滤器。
- **导出过滤器**

  将当前过滤器导出到文件中。
- **验证脚本**

  测试当前所选过滤器规则是否有效。
- **上移规则**

  交换规则列表中的当前所选规则和其上面规则的位置。
- **下移规则**

  交换规则列表中的当前所选规则和其下面规则的位置。

2.5.2 规则列表

显示规则列表中从 0 开始的规则索引。

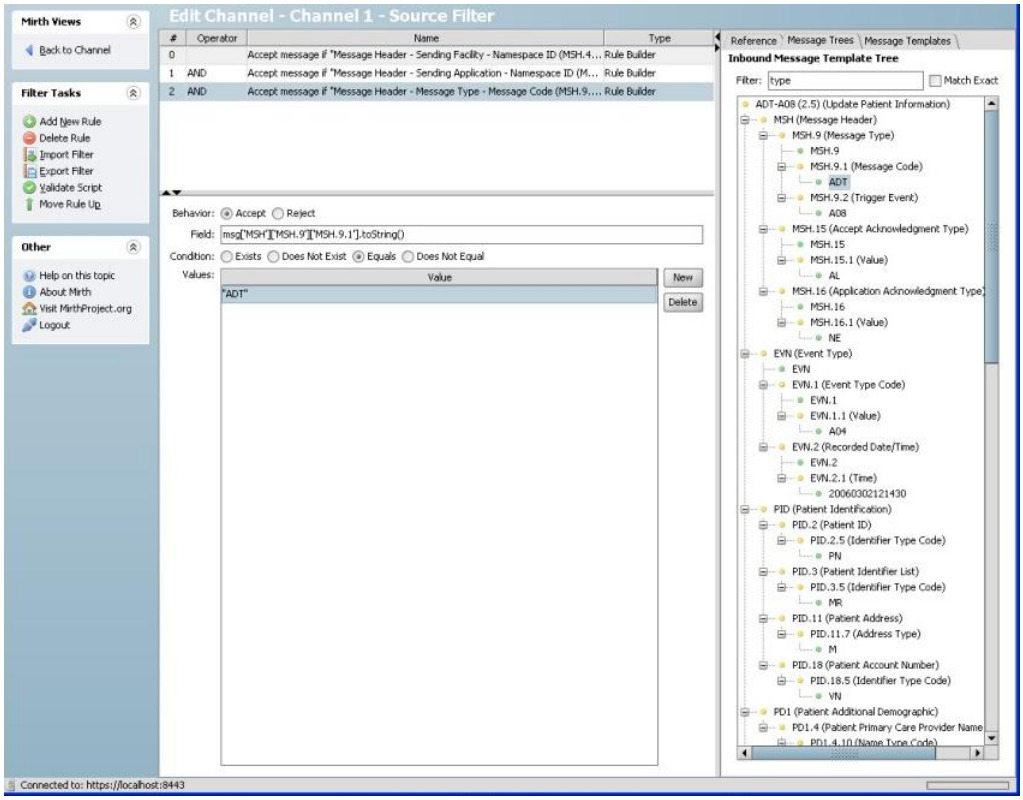
- **操作**

  显示规则列表中连接当前规则和其上条规则的布尔运算符：AND 或 OR。双击此列选择操作。
- **名称**

  当规则是 Rule Builder 类型，该名称显示规则细节的只读生成描述。当规则是 JavaScript 类型，该名称显示输入的规则名称，并可双击修改该名称。
- **类型**

  显示用于创建 Rule Builder 规则或 JavaScript 规则的方法。双击此列选择规则类型。更改规则类型删除规则细节。

2.5.3 规则细节（Rule Builder 类型）



当规则类型是 Rule Builder 时，将显示以上规则细节区域。

- **行为（接受/拒绝）**

  若选择接收，如果消息条件得到满足该规则将接收消息。反之若选择拒绝，如果消息条件得到满足该规则将拒绝消息。

- **字段**

  E4X 表达式选择消息的一部分由规则进行测试。通常是将消息树中的节点拖动到编辑控件中生成的。

- **条件**

  规则测试的条件。

存在	如果指定字段出现在所有消息中，那么规则条件为真。这种情况下不使用值列表。
不存在	如果指定字段不在消息中，那么规则条件为真。这种情况下不使用值列表。
相等	如果指定字段的值是值列表中任意一个值，那么规则条件为真。
不相等	如果指定字段的值与值列表中任意一个值都不相同，那么规则条件为真。

- **值**

如果条件为等于或不等于，将显示值列表与指定消息字段进行比较。该字段为 JavaScript 类型，因此，如果输入一个文字字符串，必须用引号括起来。

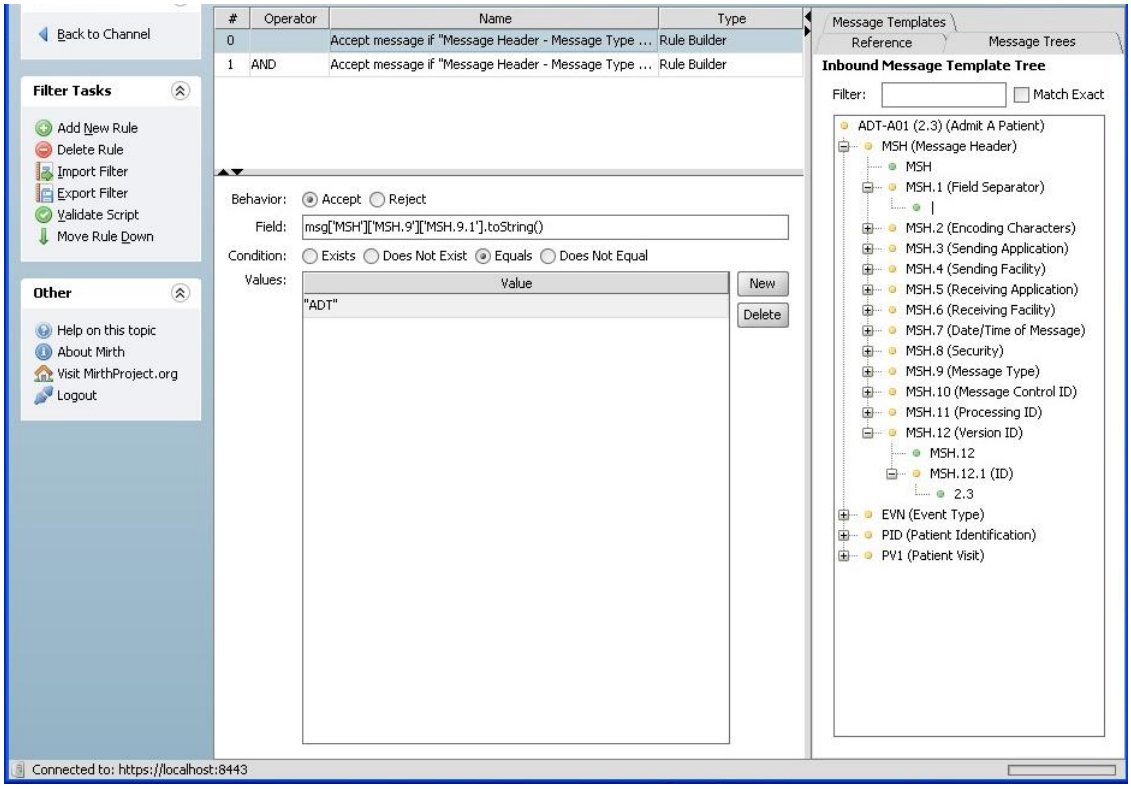
- **新增**

  在值列表中添加新的行。双击新行键入要测试的值（如果有必要加引号），并按回车键。

- **删除**

  删除值列表中当前选中的行。

2.5.4 规则细节（JavaScript 类型）

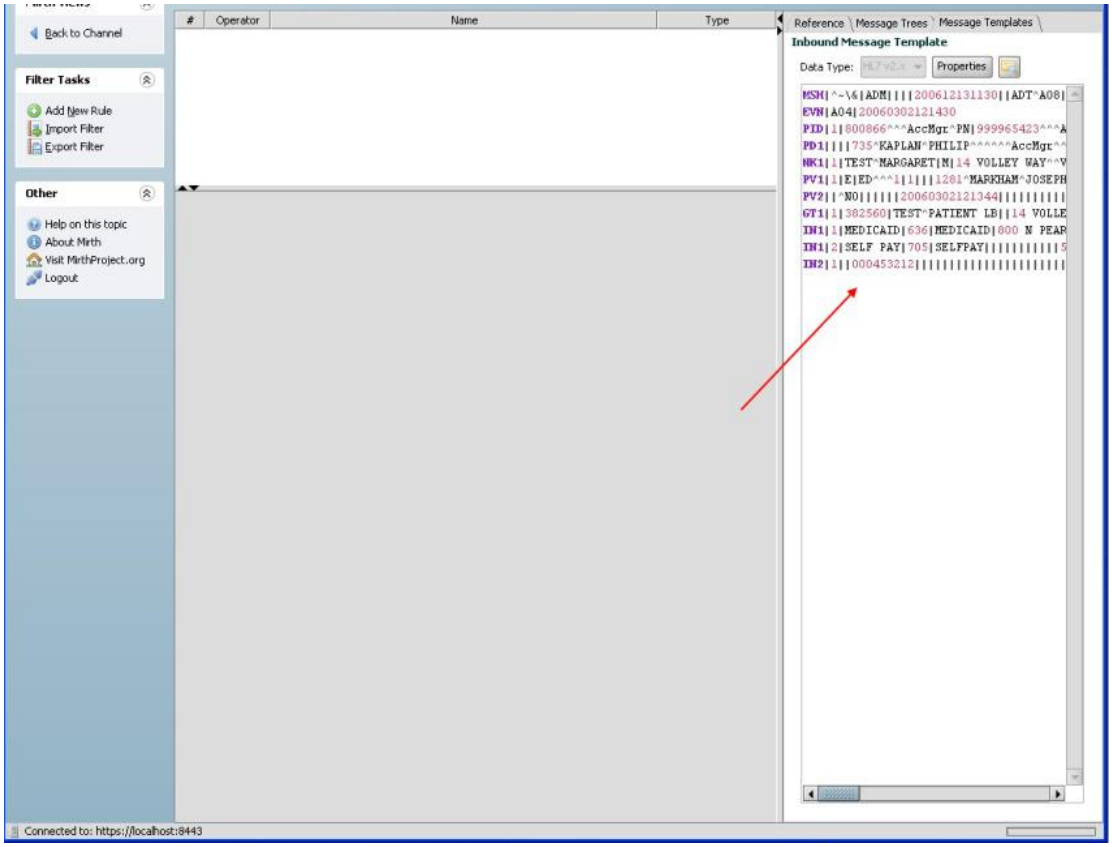


当规则类型是 JavaScript 时，将显示以上规则细节区域。

- **JavaScript 编辑器**

编辑区域是有语法色彩的 JavaScript 编辑器。从参考和消息树标签中拖动元素来构造所需的规则。JavaScript 规则会返回“真”接收消息，或者返回“假”拒绝消息。此外，右击文本区域获取更多选项，如显示行结束字符和查找/替换。

2.5.5 消息模板标签（过滤器）

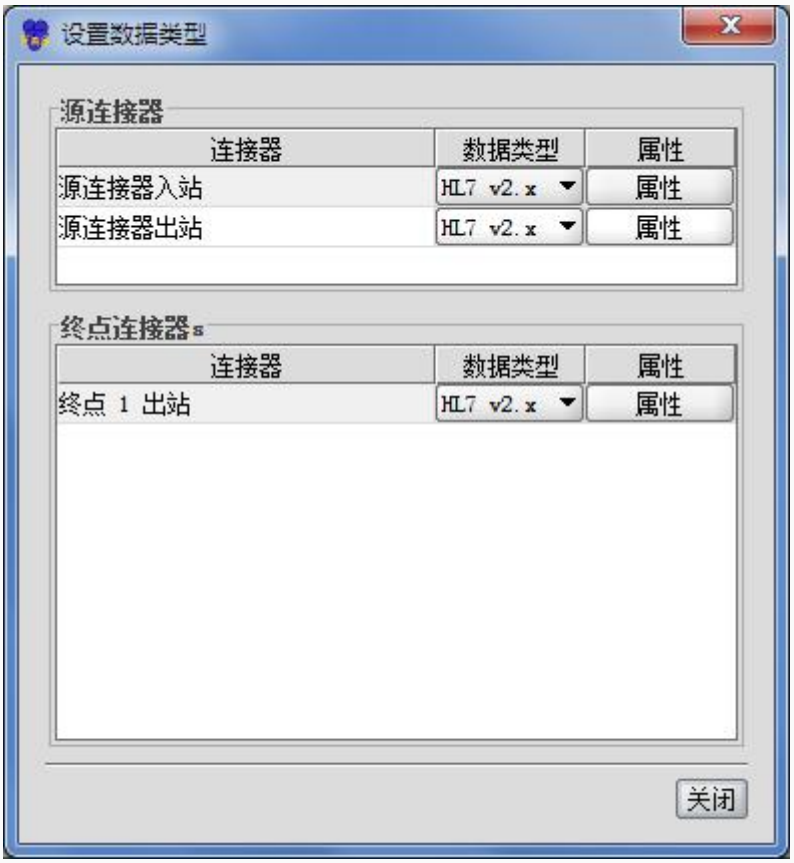


- **数据类型**

  在这里显示期望传入的消息格式。可以在通道编辑器的摘要标签定义消息格式。

- **属性（数据类型为 HL7 v2.x）**

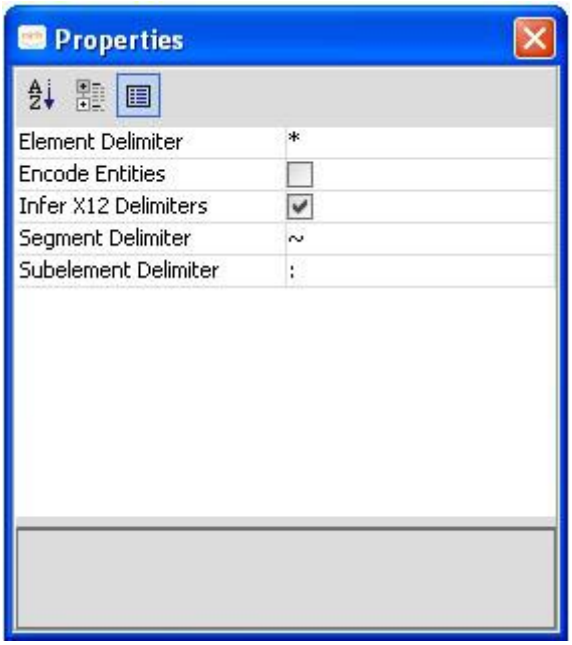
  当消息格式为 HL7 v2.x 时，点击“属性”按钮将弹出以下对话框，指定通道期望的消息格式的选项细节。



名称	描述
处理重复	如果由解析器处理消息段内的重复时选中此复选框。
使用严格解析器	默认情况下，医易通使用解析器解析 HL72.x 消息，并容忍一些非标准消息。如果医易通使用与 HL72.x 标准相匹配的解析器时选中此复选框。
在严格解析器进行验证	如果传入消息基于严格解析器是有效的，选中此复选框来检查传入消息。
转换 LF 为 CR	在解析消息之前，选中此复选框将 LF (换行)字符转换成 CR(回车)字符。这是默认选中的。

- **属性（数据类型为 X12）**

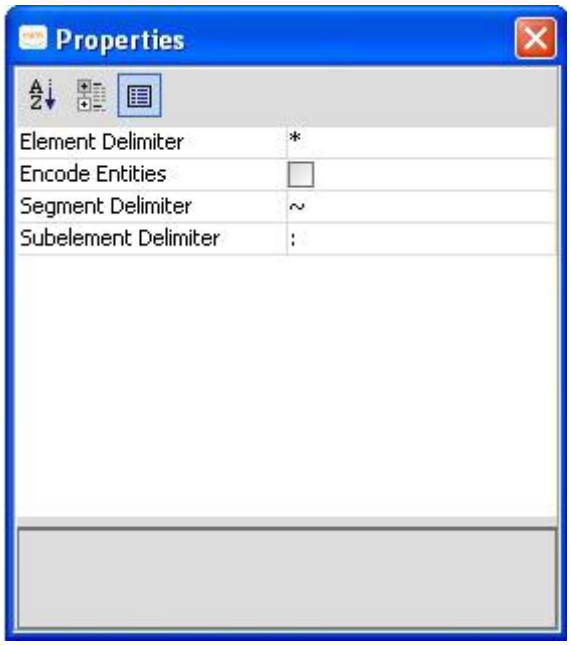
  当消息格式为 X12 时，点击“属性”按钮将弹出以下对话框，指定通道期望的消息格式的选项细节。



名称	描述
元素分隔符	如果没有选中推断 X12 分隔符，元素之间使用该分隔符。
推断 X12 分隔符	选中此复选框允许医易通通过检查消息来确定 X12 分隔符。不选中则手动指定分隔符。
段分隔符	如果没有选中推断 X12 分隔符，段之间使用该分隔符。
子元素分隔符	如果没有选中推断 X12 分隔符，子元素之间使用该分隔符。

- **属性（数据类型为 EDI）**

  当消息格式为 EDI 时，点击“属性”按钮将弹出以下对话框，指定通道期望的消息格式的选项细节。

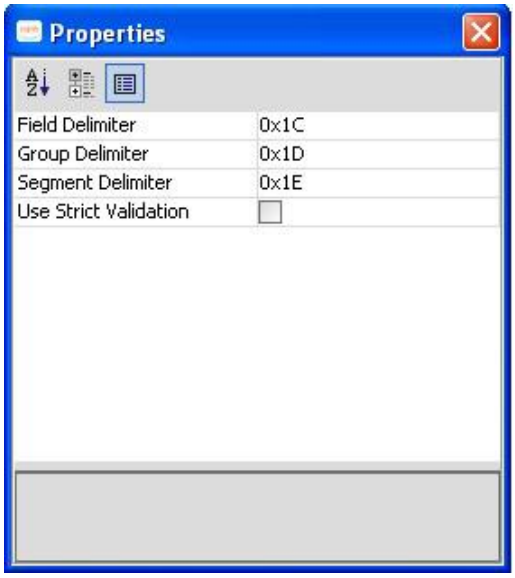


名称	描述
元素分隔符	元素之间的分隔符
段分隔符	段之间的分隔符
子元素分隔符	子元素之间的分隔符

- **属性（数据类型为 NCPDP）**

  当消息格式为 NCPDP 时，点击“属性”按钮将弹出以下对话框，指定通道期望的消息格式的选项细节。

名称	描述
字段分隔符	字段之间的分隔符
组分隔符	组之间的分隔符
段分隔符	段之间的分隔符
使用严格验证	选中此复选框，拒绝不完全匹配 NCPDP 标准的消息。不选中此框则接受不完全符合标准的消息，但仍然可以解释。



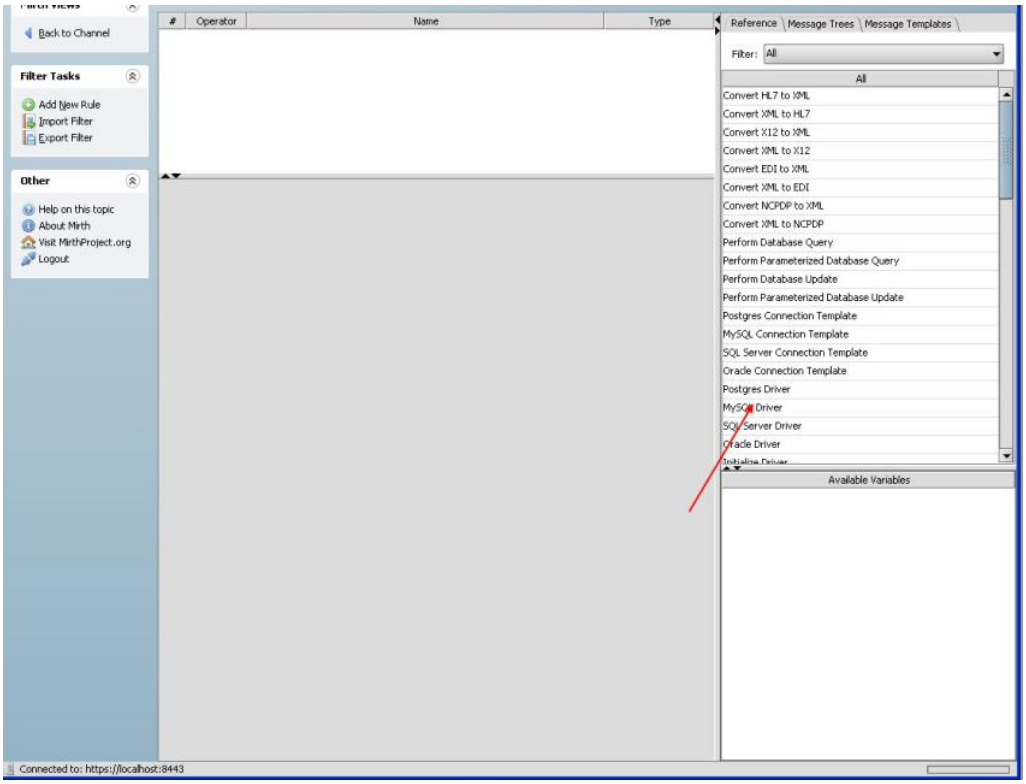
- **打开文件**

  点击此按钮打开标准的“打开文件”对话框，选择一个消息入加载到入站消息模板控件。这对于加载二进制数据模板特别有用。

- **入站消息模板**

  此编辑控件包含一个解析到“消息树”标签中的消息。使用“打开文件”按钮，可以直接输入、粘贴、或加载消息。此外，右击文本区域获取更多选项，如显示行结束字符和查找/替换。

2.5.6 参考标签（过滤器）



参考选项卡显示 JavaScript 模板和可用变量的列表，可用变量可以拖放到 JavaScript 编辑器中。

- **过滤器**

  从过滤器控件中选择映射类型，来显示下面的参考列表中该类型的映射。

- **参考列表**

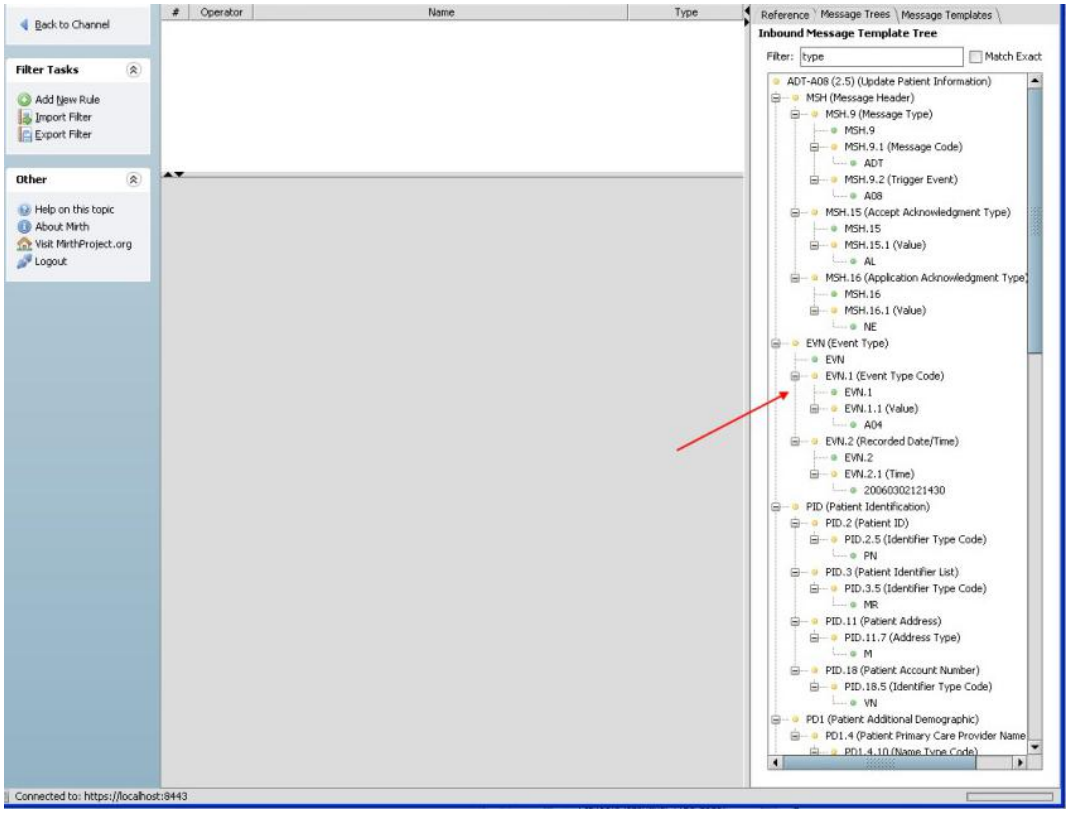
  显示可以拖放到 JavaScript 编辑器中的可用 JavaScript 模板（片段）列表。模板列表取决于选定的过滤器。

  参阅参考列表章节获取更多细节信息。

- **可用变量**

  显示可以拖放到 JavaScript 编辑器中的可用变量列表。显示的变量列表是上下文敏感的，也就是说，只显示规则可以合法参考的变量。

2.5.7 消息树标签（过滤器）



“消息树”选项卡显示输入到消息模板选项中的消息的解析和注释形式，并允许拖动和放置部分解析消息的参考到 JavaScript 规则编辑器或规则生成器中。

- **过滤器**

  在这里输入一个字符串过滤消息树只显示节点，包括节点文本中任何地方的输入字符串。这可以包括节点名称、节点值和节点描述。

- **精确匹配**

  选中此复选框，只显示与输入过滤器字符串相匹配的节点，作为相同的情况下的整个字。不选中此复选框，则显示包含输入过滤器字符串的节点，比如一个字的一部分。

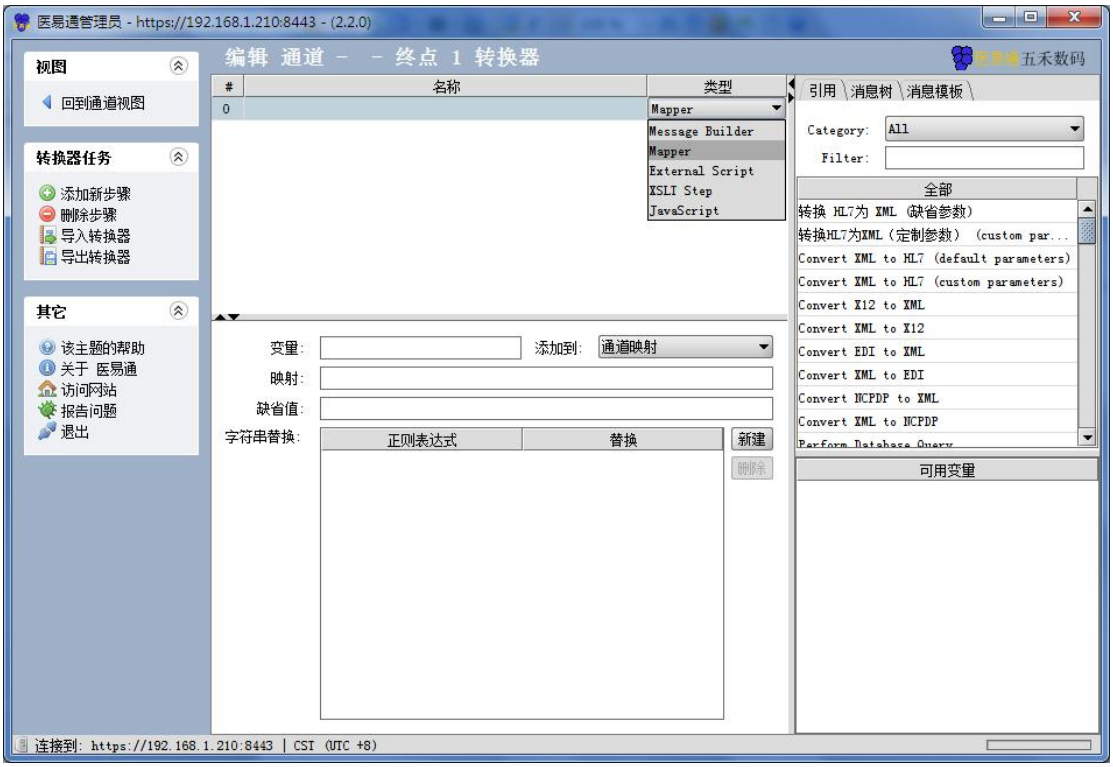
- **入站消息模板树**

  显示解析消息树的过滤节点。这些节点的参考可以拖放到 JavaScript 规则编辑器或规则构建器的字段控件中。

2.6 转换器

2.6.1 转换器任务

- **添加新步骤**



在步骤列表末尾添加新的模板步骤。

- **删除步骤**

从步骤名单中删除当前选中的步骤。

- **导入转换器**

用保存转换器文件的内容替换整个转换器，包括所有步骤。

- **导出转换器**

复制整个转换器，包括所有步骤，到保存转换器文件中。

- **验证脚本（仅 JavaScript 步骤类型）**

测试是否是 JavaScript 步骤编译。

- **上移步骤**

在步骤列表中交换所选步骤和其上一个步骤的位置。

- **下移步骤**

在步骤列表中交换所选步骤和其下一个步骤的位置。

2.6.2 步骤列表

显示步骤列表中步骤从零开始的索引。

- **名称**

对于映射步骤，显示步骤创建的变量的名称。对于消息生成器和 JavaScript 的步骤，显示可选步骤名称，并且可以通过双击选择编辑。

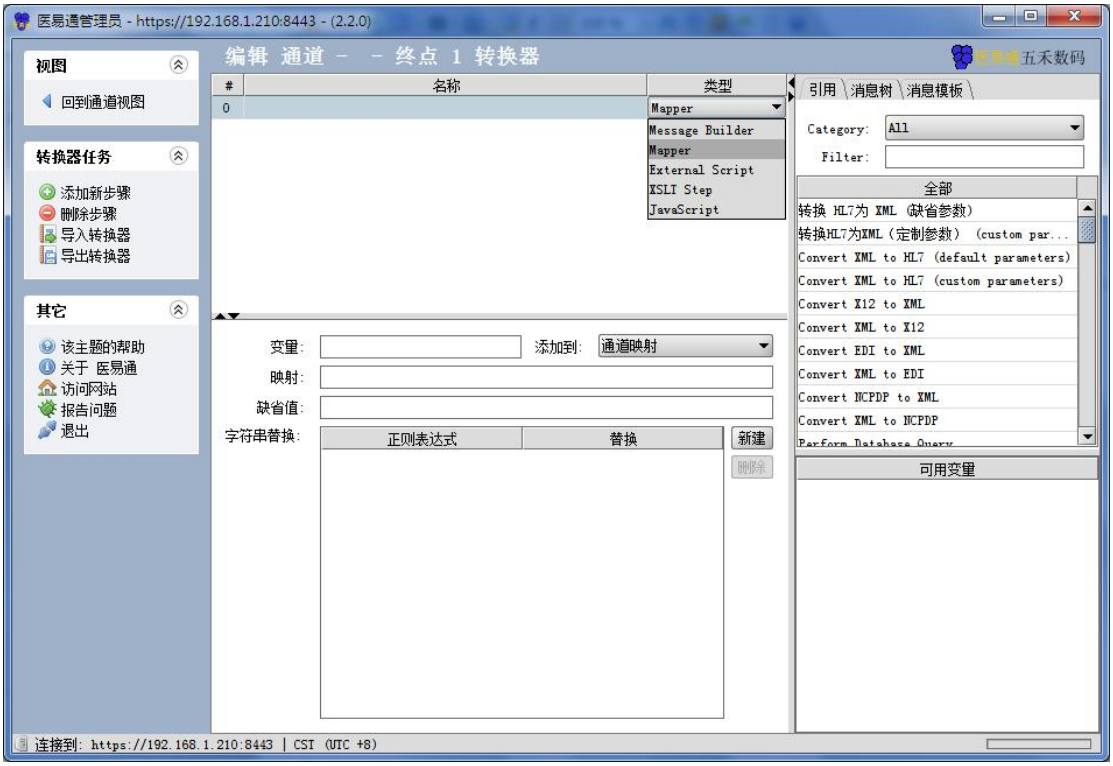
- **类型**

显示步骤类型：映射、消息生成器，或 JavaScript。双击此列可选择步骤类型。

2.6.3 步骤细节（消息生成器类型）

消息生成步骤将一个值从入站消息，转移到入站或出站消息处，转移中可能要执行一些转换。

当步骤为消息生成器类型，将出现以上步骤的详细信息。



- **消息段**

E4X 表达式指定被取代的输出消息的一部分。通常，可以从出站消息模板树中拖放表达式。

- **映射**

  被替换到输出消息段中的值。这是一个典型的 E4X 表达式用于指定输入消息的字段中。

- **默认值**

  输入消息没有映射时保存在出站消息段中的值。该字段允许为 JavaScript 类型，因此，如果输入一个文字字符串，应该用引号括起来。

- **字符串替换**

  在映射值或默认值存储到消息段之前，可以使用字符串替换表翻译字符串。对于字符串替换列表的每一行，正则表达式要与映射值或默认值进行比较。如果值匹配，则被替换为替换值，并且用之前替换的结果评估其他的行。如果没有匹配的行，将存储原来的映射或默认值。

- **新增**

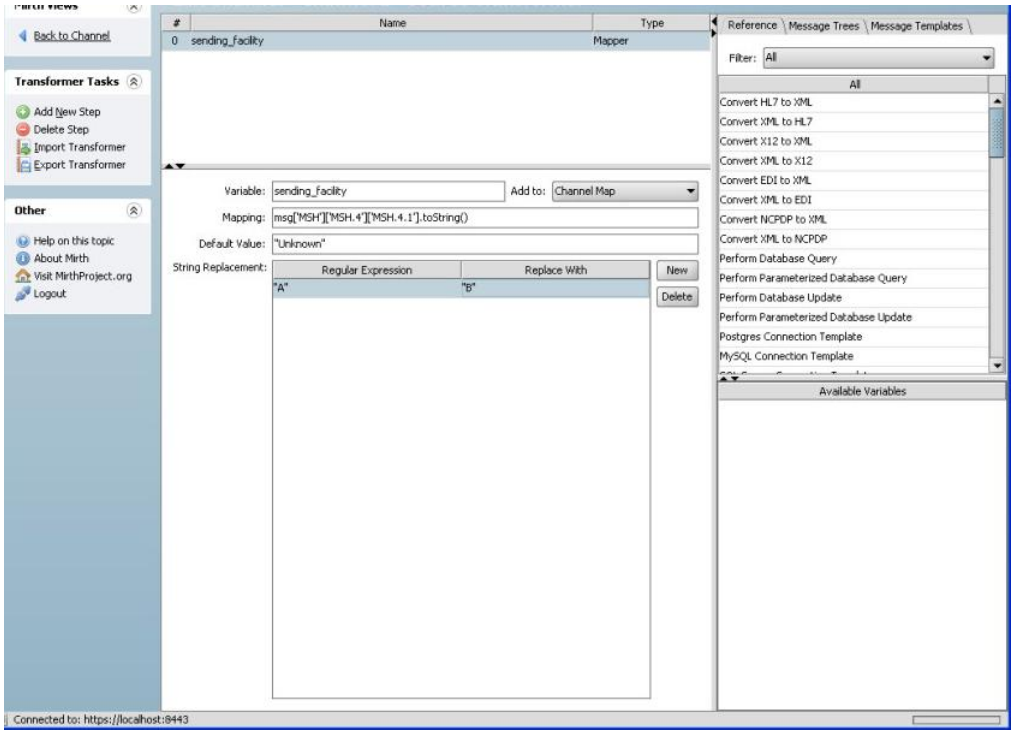
  在字符串替换列表的末尾添加一个新行。双击正常表达式单元输入新行，并按回车键接受输入值。双击替换单元输入新行并按回车键接受输入值。

- **删除**

  删除字符串替换列表中选中的行。

2.6.4 步骤细节（映射类型）

映射步骤从入站消息提取一个值给一个变量，这过程可能执行一些转换。



当步骤为映射类型时，将显示以上步骤细节。

- **变量**

  要创建的变量名。变量名只能包含字母、数字字符和下划线。

- **添加**

  选择变量将持续保存的上下文。

连接器映射	在这种情况下创建的变量只对当前连接器的转换器的后续步骤可见。源上定义的变量在任何目的地都不可见，反之亦然。
通道映射	在这种情况下创建的变量只对该通道此步骤后的源转换步骤、所有目的地过滤器、转换器和写入者可见。变量只对单一消息持续可用。源接收的每个消息包括它自己的通道映射内容。
全局映射	在这种情况下创建的变量对所有通道的所有过滤器、转换器、读出者和写入者都可见，并在所有入站消息总可用。
响应映射	目的地自动使用生成的相关响应填充响应映射。响应映射与通道映射具有相同的可见性，只对单一消息持续可用。

- **映射**

  E4X 表达式描述入站消息的一部分，作为该变量的源值。

- **默认值**

  如果入站消息不包含映射指定的部分，将 JavaScript 表达式作为变量源值。

- **字符串替换**

  在将映射的源值或默认值存储在变量中之前，该字符串可以基于字符串替换表被替换。对于字符串替换表中的每一行，如果源值与正则表达式相匹配，源值将被替换值替换，并不会检查更多的行。如果没有行与源值相匹配，将不变的源值存储在变量中。

- **新增**

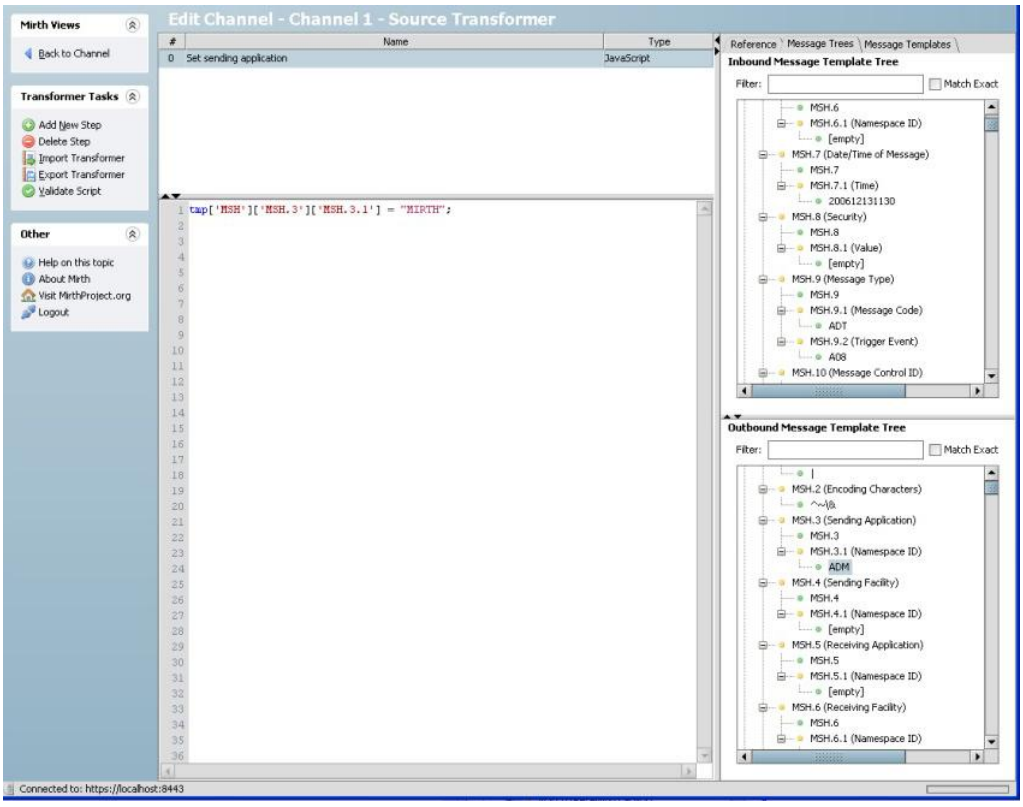
  在字符串替换表末尾添加新的行。双击正常表达单元输入新行并按回车键接受输入值。双击替换单元输入新行并按回车键接受输入值。

- **删除**

  删除字符串替换表格中选中的行。

2.6.5 步骤细节（JavaScript 类型）

JavaScript 步骤在单一的步骤中可以执行广泛的、复杂的消息格式化，进行数据库查询值转换，并执行其他行动完全无关的消息转换。

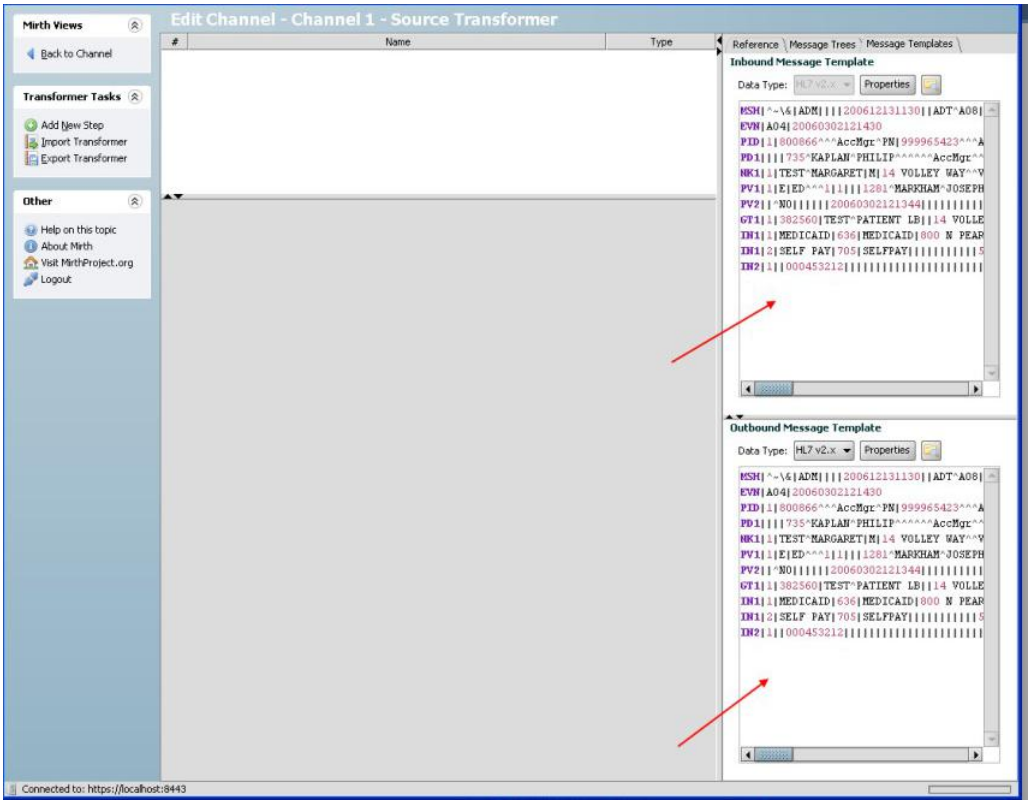


当步骤为 JavaScript 类型时，将显示以上步骤细节。

- **JavaScript 编辑器**

编辑器区域是一个有语法彩色的 JavaScript 编辑器。同时，右键单击文本区域获取更多选项，比如显示行结束符和查找/替换。从参考和消息树标签拖动元素以构建所需的步骤。这个步骤可以检查或修改变量“msg”，它包括入站消息。如果没有设置出站消息模板，这会变成了编码数据(维持对“msg”对象的所有更改)。这个步骤也可以修改个变量“tmp”，它包括出站消息。注意，在转换器开始转换时，一起加载出站消息与出站消息模板控件的内容。

2.6.6 消息模板标签（转换器）



- **入站消息模板数据类型**

在这里显示预期传入的消息格式。该消息协议是在通道编辑器摘要标签中被定义的。

- **入站消息模板属性**

  点击该按钮会显示一个属性对话框，用来选择入站消息的详细解析选项。通过属性(数据类型= NCPDP)部分查看属性(数据类型= HL7 v2 . x)部分了解详情。

- **入站消息模板打开文件**

  点击该按钮打开标准“打开文件”对话框，选择一个文件加载到入站消息模板控件中。这是特别有用的加载二进制消息的模板。

- **入站消息模板**

  该编辑控件包含解析成消息树标签的入站消息模板树控件的消息。使用“打开文件”按钮可以直接输入、粘贴、或加载消息。同时，右键单击文本区域获取更多选项，比如显示行结束符和查找/替换。

- **出站消息模板数据类型**

  选择出站消息协议。

- **出站消息模板属性**

  点击此按钮会显示一个属性对话框，为出站消息选择详细解析选项。通过属性(数据类型= NCPDP)部分查看属性(数据类型= HL7 v2 . x)部分了解详情。

- **出站消息模板打开文件**

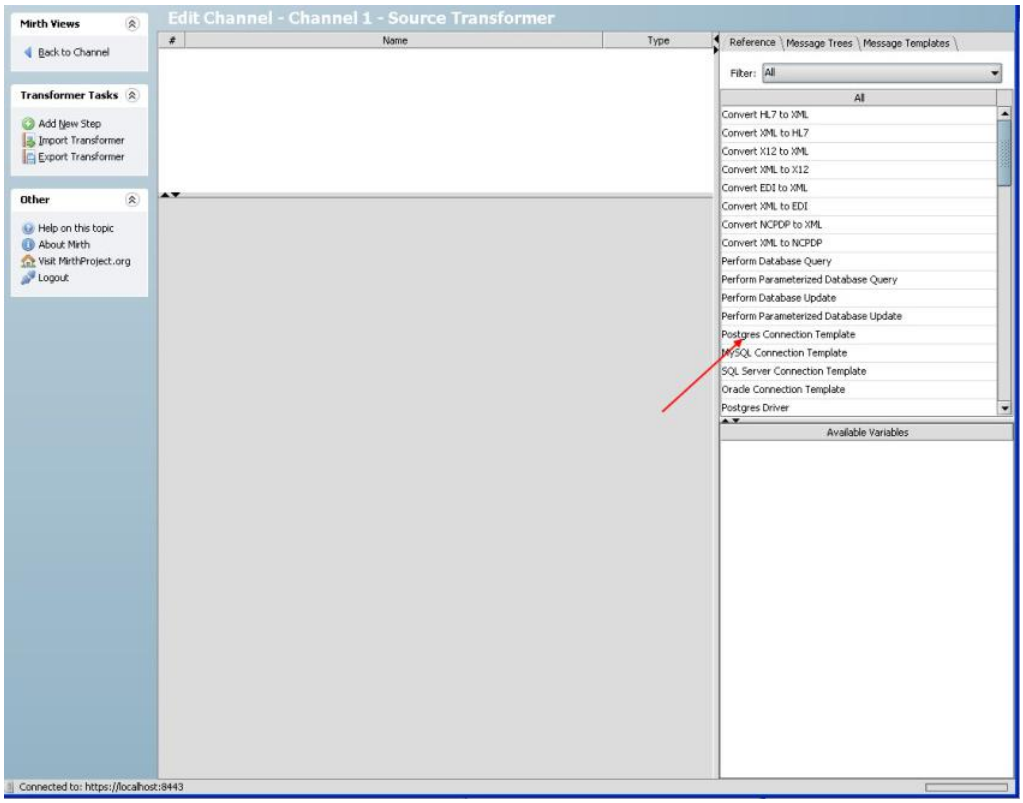
  点击此按钮打开一个标准“打开文件”对话框，来选择一个文件加载到出站消息模板控制。这是特别有用的加载二进制消息的模板。

- **出站消息模板**

  该编辑控件包含解析成消息树标签的出站消息模板树控件的消息。使用“打开文件”按钮可以直接输入、粘贴、或加载消息。同时，右键单击文本区域获取更多选项，比如显示行结束符和查找/替换。

2.6.7 参考标签（转换器）

参考标签显示 JavaScript 模板和可用变量的列表，可用变量可以拖放到 JavaScript 编辑器中。



- **过滤器**

  在过滤器控件中选择映射类型，在下面的参考列表中只显示该类型的模板。

- **参考列表**

  显示可用的 JavaScript 模板，可以拖放到 JavaScript 编辑器中。模板列表的显示取决于所选的过滤器。

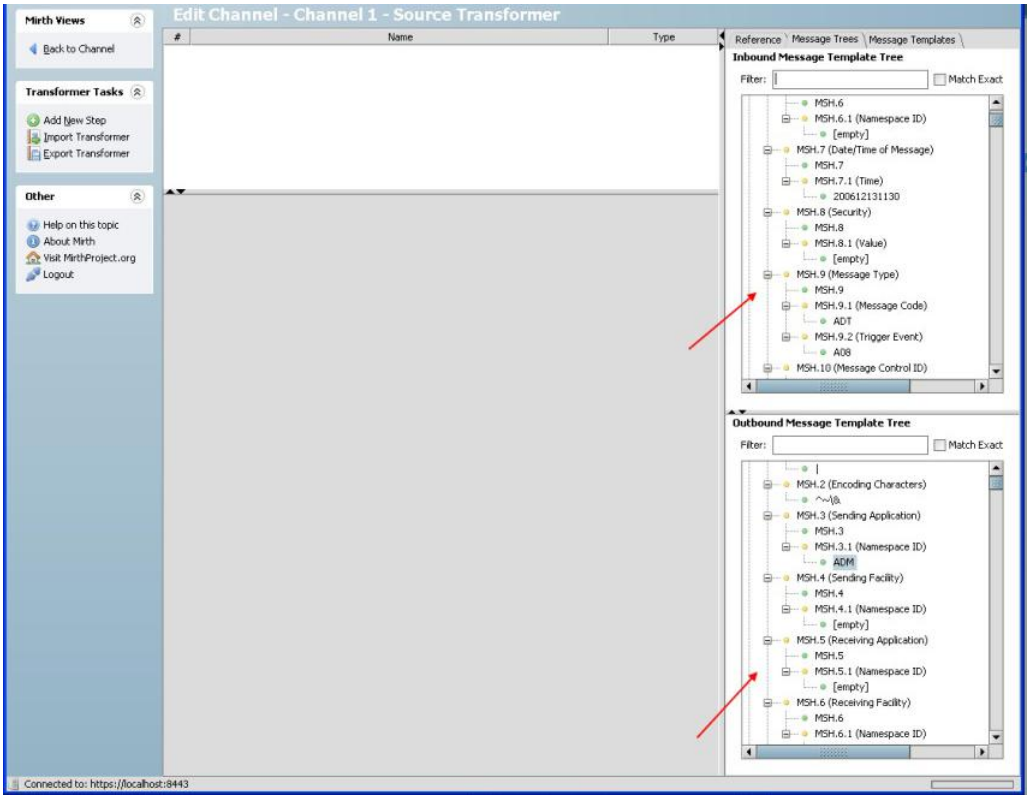
  参见参考列表章节获得更全面的细节信息。

- **可用变量**

  显示可用变量列表，可以拖放到 JavaScript 编辑器中。变量列表的显示是上下文敏感的，也就是说，可以参考规则只显示合法的变量。

2.6.8 变消息树标签（转换器）

消息树标签显示输入到消息模板标签的入站和出站消息已解析的和注释的表单，并允许拖放参考部分解析消息到消息生成器步骤的消息段控件中，映射控制消息生成器、映射步骤，或 JavaScript 步骤的 JavaScript 编辑器。



- **入站消息模板树过滤器**

  在这里输入一个字符串来过滤入站消息树，在节点文本的任何地方只显示包括输入字符串的节点。包括节点名、节点值和节点的述。

- **入站消息模板树精确匹配**

  选中此复选框，仅显示与输入过滤器字符串相匹配的入站消息树节点，作为整体字。不选中则显示包含输入过滤器字符串的节点，比如一个字的一部分。

- **入站消息模板树**

  显示解析的入站消息树的过滤节点。这些节点的参考可以拖放到消息生成器、映射步骤或 JavaScript 步骤的 JavaScript 编辑器的映射控制中。还有支持拖拽入站消息树节点的参考到消息生成器步骤的消息段控件中，以修改入站消息。

- **出站消息模板树过滤器**

  在这里输入一个字符串来过滤出站消息树，在节点文本的任何地方只显示包括输入字符串的节点。包括节点名、节点值和节点的述。

- **出站消息模板树精确匹配**

  选中此复选框，仅显示与输入过滤器字符串相匹配的出站消息树节点，作为整体字。不选中则显示包含输入过滤器字符串的节点，比如一个字的一部分。

- **出站消息模板树**

  显示解析的出站消息树的过滤节点。这些节点的参考可以拖放到消息生成器、映射步骤或 JavaScript 步骤的 JavaScript 编辑器的映射控制中

2.7 转换器管理:

2.7.1 连接器管理:

源连接器

目标连接器

2.8 消息浏览:

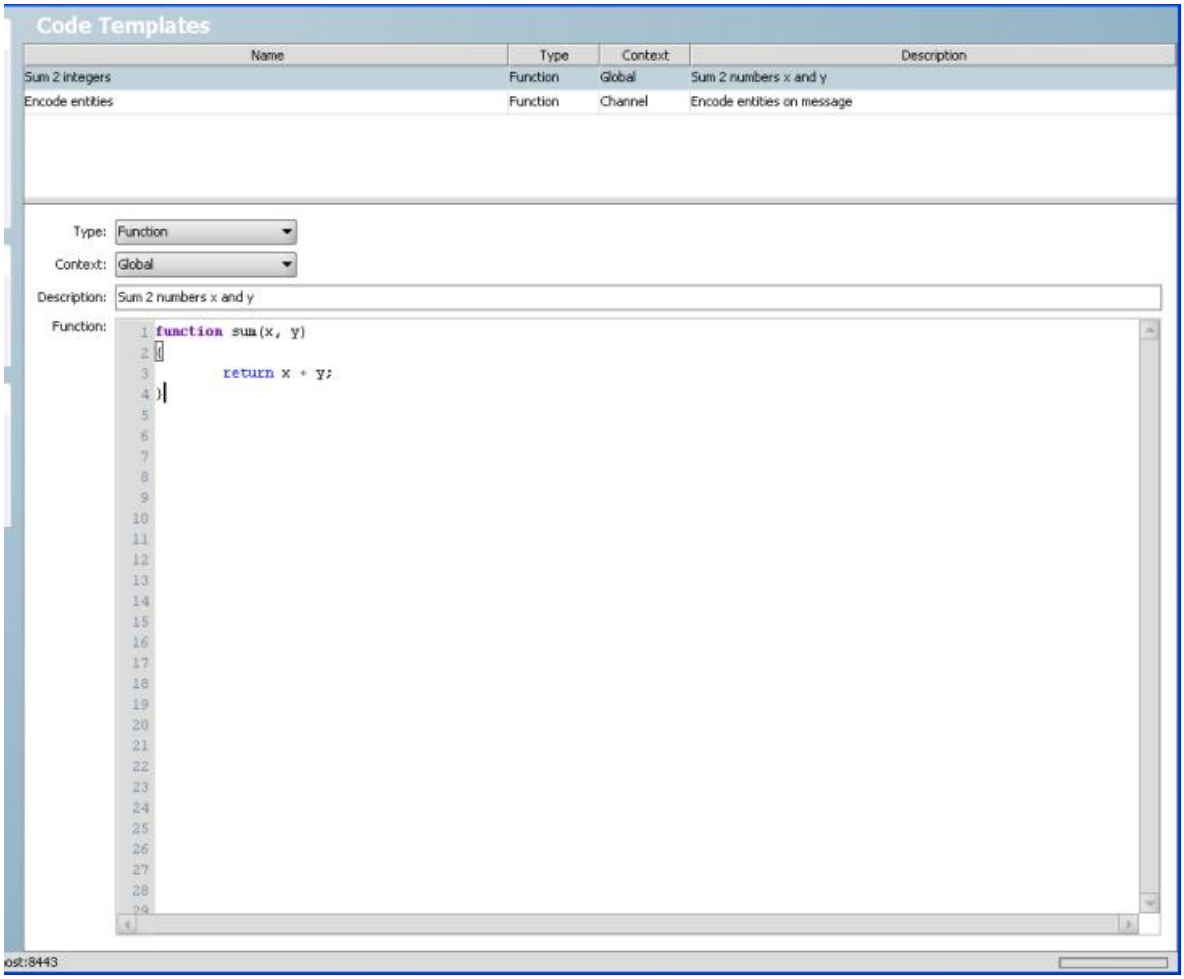
2.9 用户管理:

2.10 系统设置:

2.11 警告管理:

2.12 代码模板管理:

The Code Templates area provides tools for managing templates for variables and code along with global functions. The templates and functions that are defined show up in the reference list when editing scripts throughout Gateway .



代码模板管理任务

I'd figure I would start out with some functions that some may find useful when developing channels.

Feel free to use them, tweak them, or add more to this thread of others to gaze upon. I hope to start other threads with other tools/scripts/code examples to assist in making our daily yytConnect experience that more enjoyable and less migraine inducing.

这些函数可以在通道中定义，只能在通道内调用，或者放入代码模板部分，可以被所有通道使用。

1. **Determine a Channel's status**

Purpose: To determine the current state of a channel.

Sometimes, you need to know the status of a certain channel, from within another channel. To accomplish this, you can use the following to see if the channel is :
-STARTED
-STOPPED

-PAUSED
-NA (Non-available)

Code:

```
function GetChannelState(channel_id) {

    var channel_status = "NA";

    var channel_count =
parseInt(Packages.com.webreach.yyt.server.controllers.ChannelStatusController.g
etInstance().getChannelStatusList().size());

    for(var i=0;i<channel_count;i++) {
        if (channel_id ==
Packages.com.webreach.yyt.server.controllers.ChannelStatusController.getInstanc
e().getChannelStatusList().get(i).getChannelId()) {
            channel_status =
Packages.com.webreach.yyt.server.controllers.ChannelStatusController.getInstanc
e().getChannelStatusList().get(i).getState();
        }
    }

    return channel_status;

}
```

If you do not know your channel_id, you can view it next to the channel_name on the 'Channels' section

2. **Quick SELECT/UPDATE/INSERT functions**

Purpose: If most of your channels are connecting to the same database, you can use these functions to cut down some lines of **code** and for re-usability

Let's say that all of your channels are connecting to the same MySQL database. You could create a SELECT function as such:

Code:

```
function CDRSelect(sql) {

    var dbConn =
DatabaseConnectionFactory.createDatabaseConnection("com.mysql.jdbc.Driver","
jdbc:mysql://localhost:3306/cdrexample","USER","PASSWORD");

    var results = dbConn.executeCachedQuery(sql);
```

```
    dbConn.close();

    return results;
}
```

Now, in your Source or in a javascript transformer, you can execute a SELECT query by calling the following:

Code:

```
var results= CDRSelect("SELECT * FROM results");
```

....and for an UPDATE query, it would look like the following:

Code:

```
function CDRUpdate(sql) {

    var dbConn =
DatabaseConnectionFactory.createDatabaseConnection("com.mysql.jdbc.Driver","
jdbc:mysql://localhost:3306/cdrexample","USER","PASSWORD");
    var results = dbConn.executeUpdate(sql);
    dbConn.close();

    return results;
}
```

...which can be referenced by :

Code:

```
var results= CDRUpdate("UPDATE results set name = 'test'");
```

3. **Misc. Trim functions**

Purpose: Sometimes, you need to do some trimming of strings. If you do not feeling like memorizing Regex, you can use something along the lines of the following.

Code:

```
function trim(in_str) {
    return new String(in_str).replace(/^\s+|\s+$/g,"");
}
```

Code:

```
function ltrim(in_str) {
    return new String(in_str).replace(/^\s+/,"");
```

```
}
```

Code:

```
function rtrim(in_str) {
     return new String(in_str).replace(/\s+$/,"");
}
```

Now, the function can be called within a transformer or mapper step:

Code:

```
var fname= trim(msg['patient_first'].toString());
```

5: **Local Timzones**

Purpose: Sometimes, your not running yytlocally and you need the correct timezone.

Code:

```
function getCurrentLocalTimestamp() {
    var formatter = new
Packages.java.text.SimpleDateFormat("yyyyMMddhhmmss");
    // your local TZ
    formatter.setTimeZone(Packages.java.util.TimeZone.getTimeZone("EST"));
    return formatter.format(new Packages.java.util.Date());
}
```

The function can be used to get the local server date/time by the following:

Code:

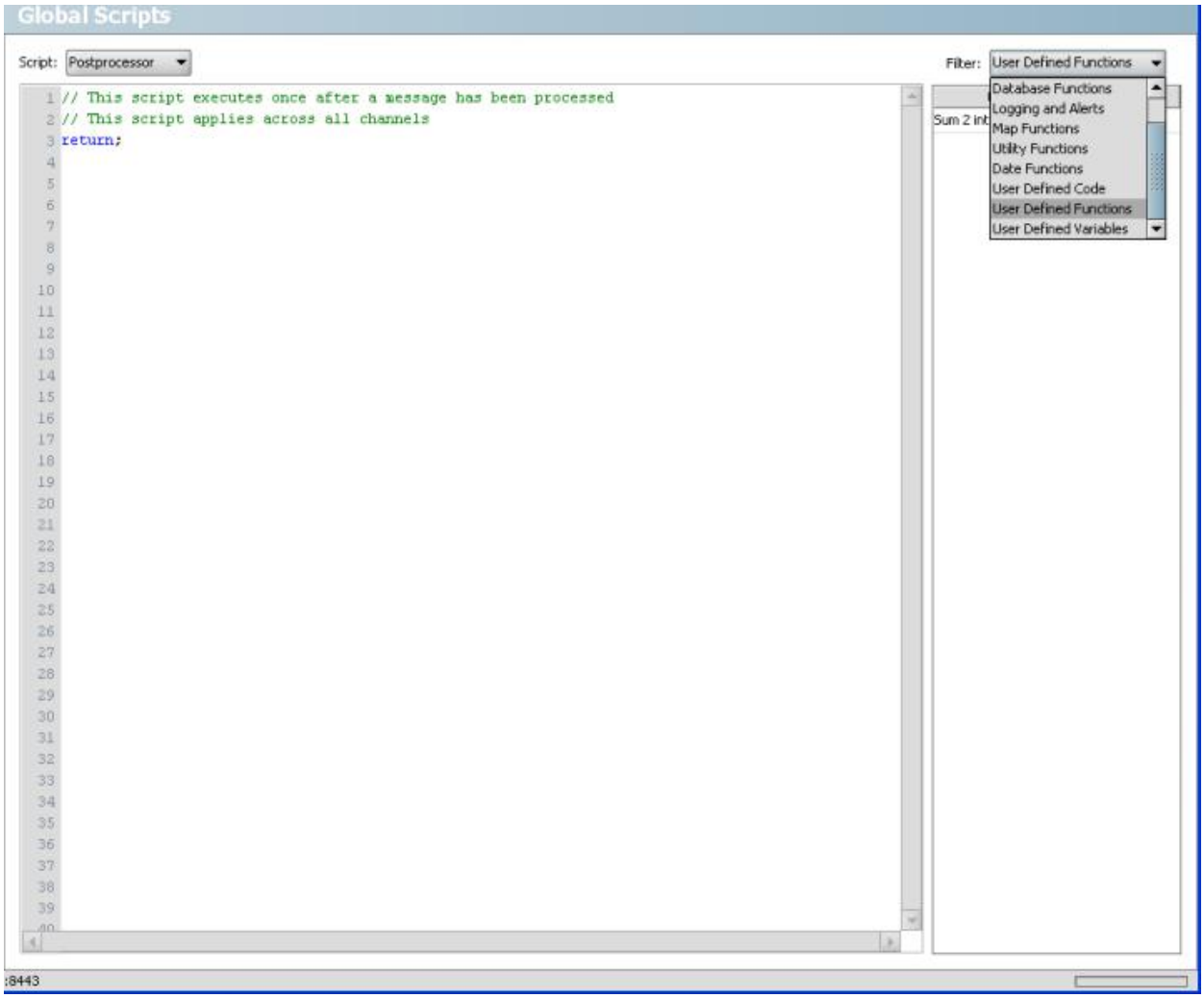
```
tmp['MSH']['MSH.7']['MSH.7.1'] = getCurrentLocalTimestamp();
```

Code Template List

Code Template Panel

引用列表

After defining code templates, check the reference lists in the various script editors and drag and drop the templates to use them. There will be a section corresponding to each of the types of templates prefixed with the words "User Defined." Right click on a scripting text box to see the same templates; however, these are only updated when the 管理员 is first launched.



下面是 过滤了的可用 JavaScript templates 简要列表，可以 dragged from the list and dropped into the Editor.

Utility Functions	**Description**
Use Java Class	Provides the ability to use any Java class that is on the built-in classpath or loaded from the conf/custom folder. The fully qualified path-name is required
Generate Unique ID	Generate a standard Globally Unique ID (GUID)
Call System Function	Execute any system program or function that is available on the current yyt user's path
Read File As String	Reads a file from the file system and returns the contents as a Java string
Read File As Bytes	Reads a file from the file system and returns the contents as a Java byte array

Write String to File	Write the specified Java/JavaScript string to the specified file path
Write Bytes to File	Write the specified Java byte array to the specified file path
BASE-64 Encode Data	Converts a Java byte array into a BASE-64 encoded Java string
Decode BASE-64 Data	Converts a BASE-64 encoded Java string into a Java byte array
Route Message to Channel	Sends any Java string to any channel in the current yyt instance based on the channel name.
Route Message to Channel (w/ queue, synchronized)	Sends any Java string to any channel in the current yyt instance based on the channel name. Booleans can be set for useQueue and synchronized.
Perform Map Value Replacement	Replace values in any string with values from the channel map, connector map or response map.
Format Overpunch NCPDP Number	Converts currency from the Overpunch format to standard format. For example, '999E' would be converted to '$99.95'.
Convert DICOM into Image File	Returns a JPEG/TIF image from an uncompressed DICOM image (either TIF, JPEG, BMP, PNG, or RAW).
Get DICOM message	Returns a byte array representation of the DICOM message, including any attachments.
Add Attachment	Adds attachment data to the current message object.
Get Attachments	Returns an array of all attachments in the current message object.

Message Functions	**Description**
Incoming Message	The raw message
Incoming Message (XML)	The raw message as XML
Message Type	The message type (e.g. ADT-A01)

Message Source	The sending facility that delivered the message.
Message Version	The version of the message (e.g. HL7 v2.3)
Message ID	Message ID in yyt.
Message Protocol	The message protocol (e.g. XML, HL7 v2.x)
Channel ID	The channel's ID in yyt.
Iterate Over Segment	A block of code that illustrates how to iterate over a group of segments with the same name.
Create Segment (individual)	Creates a new segment that can be used in any message.
Create Segment (in message)	Creates segment in the msg object.
Create Segment (in message, indexed)	Creates a segment in the msg object at a certain index.
Create Segment After Segment	Creates a segment after the provided segment object.
Delete Segment	Deletes a segment given its name.

Date Functions	**Description**
Get Date Object From Pattern	Returns a Java Date object from a specific date string and the Java SimpleDateFormat 1 pattern that date string is formatted in.
Format Date Object	Returns a date string from a Java Date object and a SimpleDateFormat pattern to use for formatting the date string.
Convert Date String	Returns a date string formatted using the given "out pattern". Needs an incoming date string and an "in pattern" the date string is currently formatted in. Both patterns are Java SimpleDateFormat patterns.
Get Current Date	Returns a date string with the current date in the format of the given SimpleDateFormat pattern.

Conversion Functions	Description
Convert HL7 to XML	Converts ER7 encoded HL7 to XML. Pass in Boolean values for useStrictParser, useStringValidation, and handleRepetitions. Takes an ER7 string as the 'message' parameter.
Convert XML to HL7	Converts XML-HL7 to ER7-HL7. Pass in Boolean values for useStrictParser, useStringValidation, and handleRepetitions. Takes an XML object as the 'message' parameter.
Convert X12 to XML	Converts X12 to XML. Pass in Boolean values for inferDelimiters. Takes an X12 string as the 'message' parameter.
Convert XML to X12	Converts XML to X12. Pass in Boolean values for inferDelimiters. Takes an XML object as the 'message' parameter.
Convert EDI to XML	Converts EDI to XML. Pass in String values for segmentDelim, elementDelim, and subelementDelim. Takes an EDI string as the 'message' parameter.
Convert XML to EDI	Converts XML to EDI. Pass in String values for segmentDelim, elementDelim, and subelementDelim. Takes an XML object as the 'message' parameter.
Convert NCPDP to XML	Converts NCPDP to XML. Pass in String values for segmentDelim, elementDelim, and groupDelim. Takes an NCPDP string as the 'message' parameter.
Convert XML to NCPDP	Converts XML to NCPDP. Pass in String values for segmentDelim, elementDelim, and groupDelim. Takes an XML object as the 'message' parameter.

Logging and Alerts	Description
Log an Info Statement	Logs specified INFO message to yyt.log and Wrapper.log (when using service).
Log an Error Statement	Logs specified ERROR message to yyt.log and Wrapper.log (when using service)

Send an Email	Sends email using SMTP
Trigger an Alert	Sends a message to the alerting system, which will trigger any alerts that were specified to match that message

Database Functions	**Description**
Perform Database Query	Perform any arbitrary database query against any supported JDBC database
Perform Parameterized Database Query	Perform any arbitrary database query against any supported JDBC database. Pass in a List of parameters to the query.
Perform Database Update	Perform any arbitrary database insert or update against any supported JDBC database
Perform Parameterized Database Update	Perform any arbitrary database insert or update against any supported JDBC database. Pass in a List of parameters to the query.
Postgres Connection Template	The connection string template for a PostgreSQL connection
MySQL Connection Template	The connection string template for a MySQL connection
SQL Server Connection Template	The connection string template for a SQL Server connection
Oracle Connection Template	The connection string template for an Oracle connection
Postgres Driver	The Postgres JDBC driver template
MySQL Driver	The MySQL JDBC driver template
SQL Server Driver	The MS SQL Server JDBC driver template
Oracle Driver	The Oracle JDBC driver template
Initialize Driver	Initializes a JDBC driver for use in creating database connections

Map Functions	Description
Lookup Value in All Maps($('key'))	Checks all maps on the message object for the specified key. Search begins with connector map then channel, response and global maps respectively.
Get Global Variable Map($g)	Retrieves an object from the global variable map
Put Global Variable Map	Inserts an object into the global variable map
Get Channel Variable Map	Retrieves an object from the channel variable map
Put Channel Variable Map	Inserts an object into the channel variable map
Get Connector Variable Map	Retrieves an object from the connector variable map
Put Connector Variable Map	Inserts an object into the connector variable map
Get Response Variable Map	Retrieves an object from the response variable map
Put Response Variable Map	Inserts an object into the response variable map
Put Success Response Variable	Puts a SUCCESS response in the Response Map with a given string as the message
Put Error Response Variable	Puts an ERROR response in the Response Map with a given string as the message

```
function $co(key, value) {
     if (arguments.length == 1) { return connectorMap.get(key); }
    else { return connectorMap.put(key, value); }
}
function $c(key, value) {
     if (arguments.length == 1) { return channelMap.get(key); }
    else { return channelMap.put(key, value); }
}
function $s(key, value) {
    if (arguments.length == 1) { return sourceMap.get(key); }
    else { return sourceMap.put(key, value); }
```

```
}
function $gc(key, value) {
    if (arguments.length == 1) { return globalChannelMap.get(key); }
    else { return globalChannelMap.put(key, value); }
}
function $g(key, value) {
    if (arguments.length == 1) { return globalMap.get(key); }
    else { return globalMap.put(key, value); }
}
function $cfg(key, value) {
    if (arguments.length == 1) { return configurationMap.get(key); }
    else { return configurationMap.put(key, value); }
}
function $r(key, value) {
    if (arguments.length == 1) { return responseMap.get(key); }
    else { return responseMap.put(key, value); }
}
```

```
function $co(key, value)
{
  if (arguments.length == 1) {
      return connectorMap.get(key); } else { return connectorMap.put(key, value); }
}
function $c(key, value) {
      if (arguments.length == 1) { return channelMap.get(key); }
      else { return channelMap.put(key, value); }
}
function $s(key, value) {
      if (arguments.length == 1) { return sourceMap.get(key); }
      else { return sourceMap.put(key, value); }
}
function $gc(key, value) { if (arguments.length == 1) { return globalChannelMap.get(key); }
else { return globalChannelMap.put(key, value); } }

function $g(key, value) { if (arguments.length == 1) { return globalMap.get(key); } else { return
globalMap.put(key, value); } }
```

```
 function $cfg(key, value) {
      if (arguments.length == 1) { return configurationMap.get(key); }
      else { return configurationMap.put(key, value); }
 }
function $r(key, value) {
      if (arguments.length == 1) { return responseMap.get(key); }
      else { return responseMap.put(key, value); }
}
function $(string) {
try {if(responseMap.containsKey(string)) { return $r(string); } }catch(e){}
try { if(connectorMap.containsKey(string)) { return $co(string); } } catch(e){}
try { if(channelMap.containsKey(string)) { return $c(string); } } catch(e){}
try { if(sourceMap.containsKey(string)) { return $s(string); } } catch(e){}
try{ if(globalChannelMap.containsKey(string)) { return $gc(string); } } catch(e){}
try { if(globalMap.containsKey(string)) { return $g(string); } } catch(e){}
try { if(configurationMap.containsKey(string)) { return $cfg(string); } } catch(e){}
try { if(resultMap.containsKey(string)) { return resultMap.get(string); } } catch(e){}
return ''; }
```

注意：There is no channel map available in the JavaScript Reader, because there ***is no message yet.*** The job of the JavaScript Reader is to ***create*** messages for the channel to consume (or just execute an arbitrary block of code).

The JavaScript Writer is different. It's a destination connector, so in there you *do* have access to the channel map.

If you want to pass metadata along with your messages in the JavaScript Reader, you can return a RawMessage object (or list of said objects), including a map of variables to inject into the source map. More info in the User API.

2.13 事件管理:

插件管理

3 例子

3.1 在数据库写适配器中访问消息（msg）对象

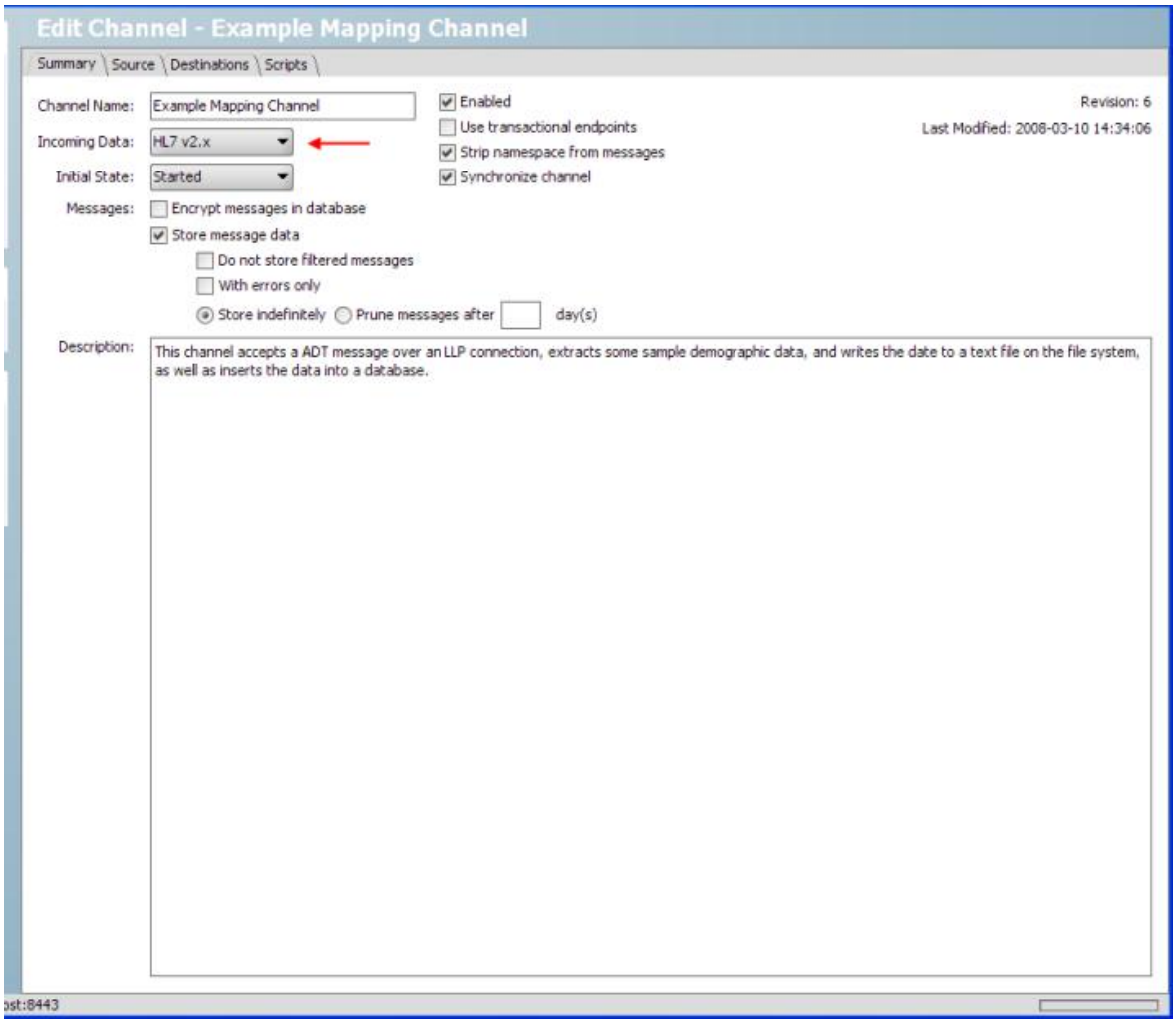
Msg 对象不能直接在数据库写者（database writer）里访问，这使得迭代消息段不可能。解决方法是从通道映射（channelMap）导入：

```
var msg = channelMap.get('msg');
```

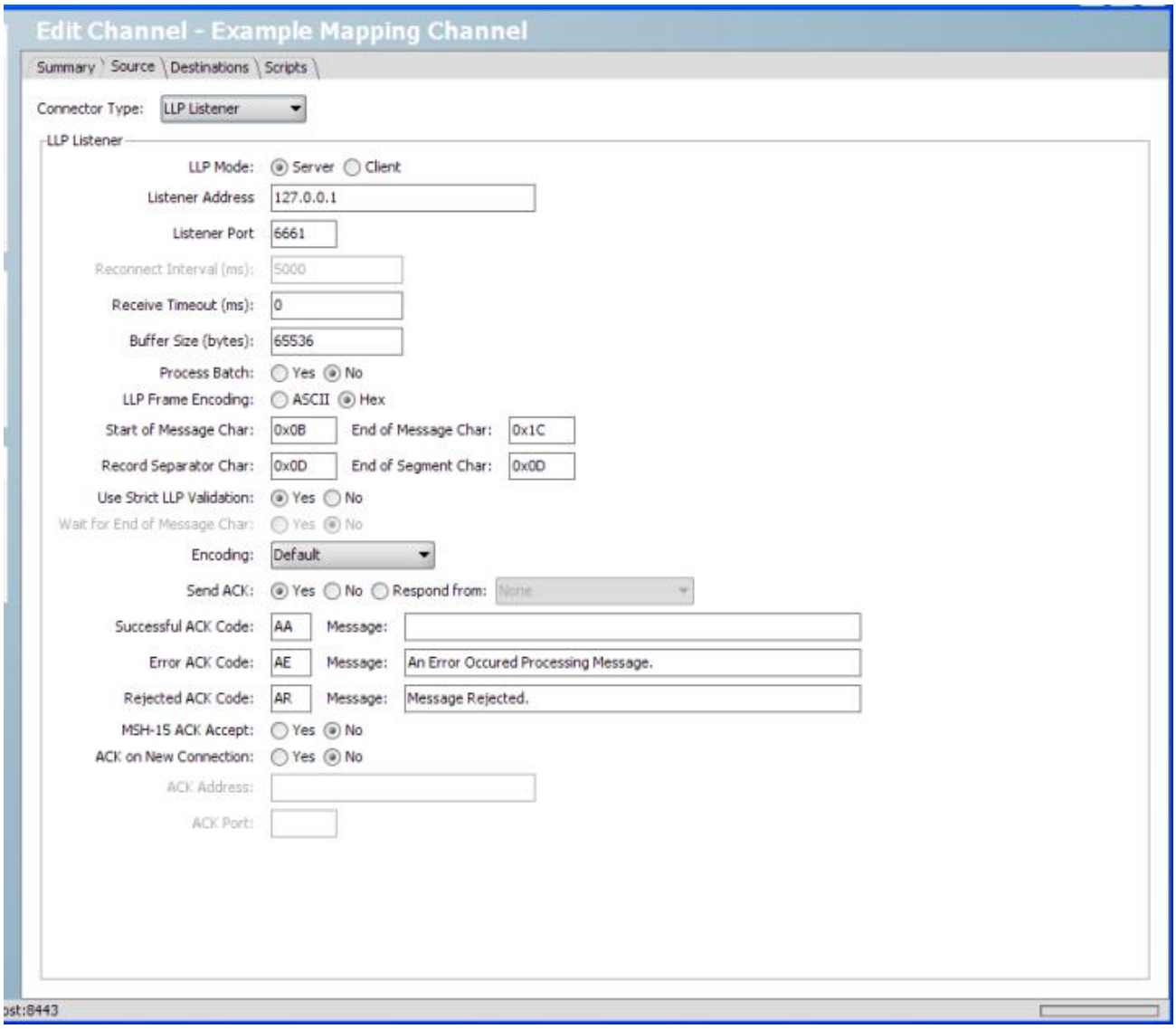
3.2 过滤器和映射转换器

this example we will configure a channel to listen for HL7 v2.x messages using and LLP connection, extract some patient data such as name and address from the message, and write the data to both a text file and a database depending on the message's sender.
This example will demonstrate:

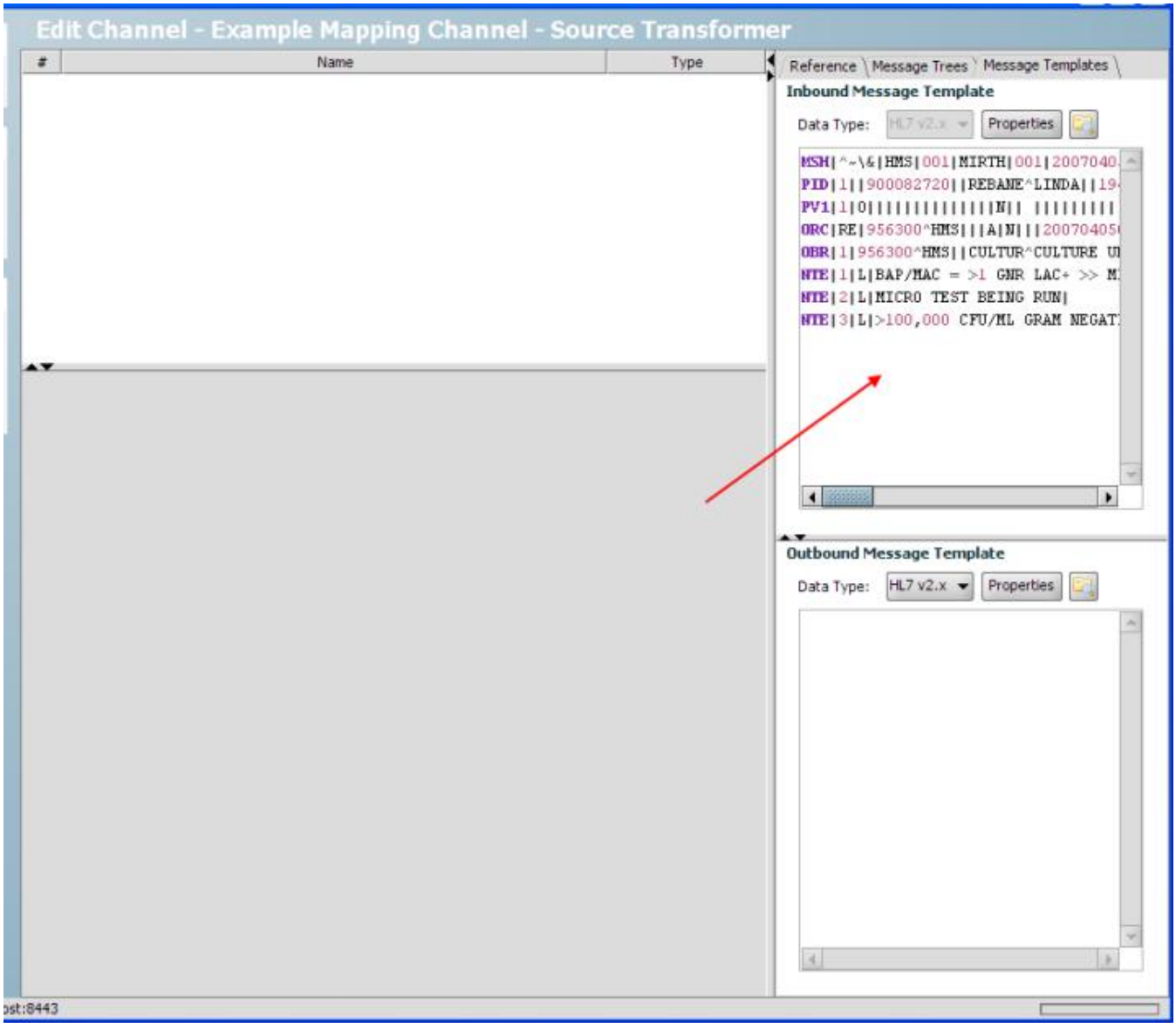
- Configuring a channel
- Using message templates
- Creating transformer mapping steps
- Creating filter rules
- Configuring multiple destinations



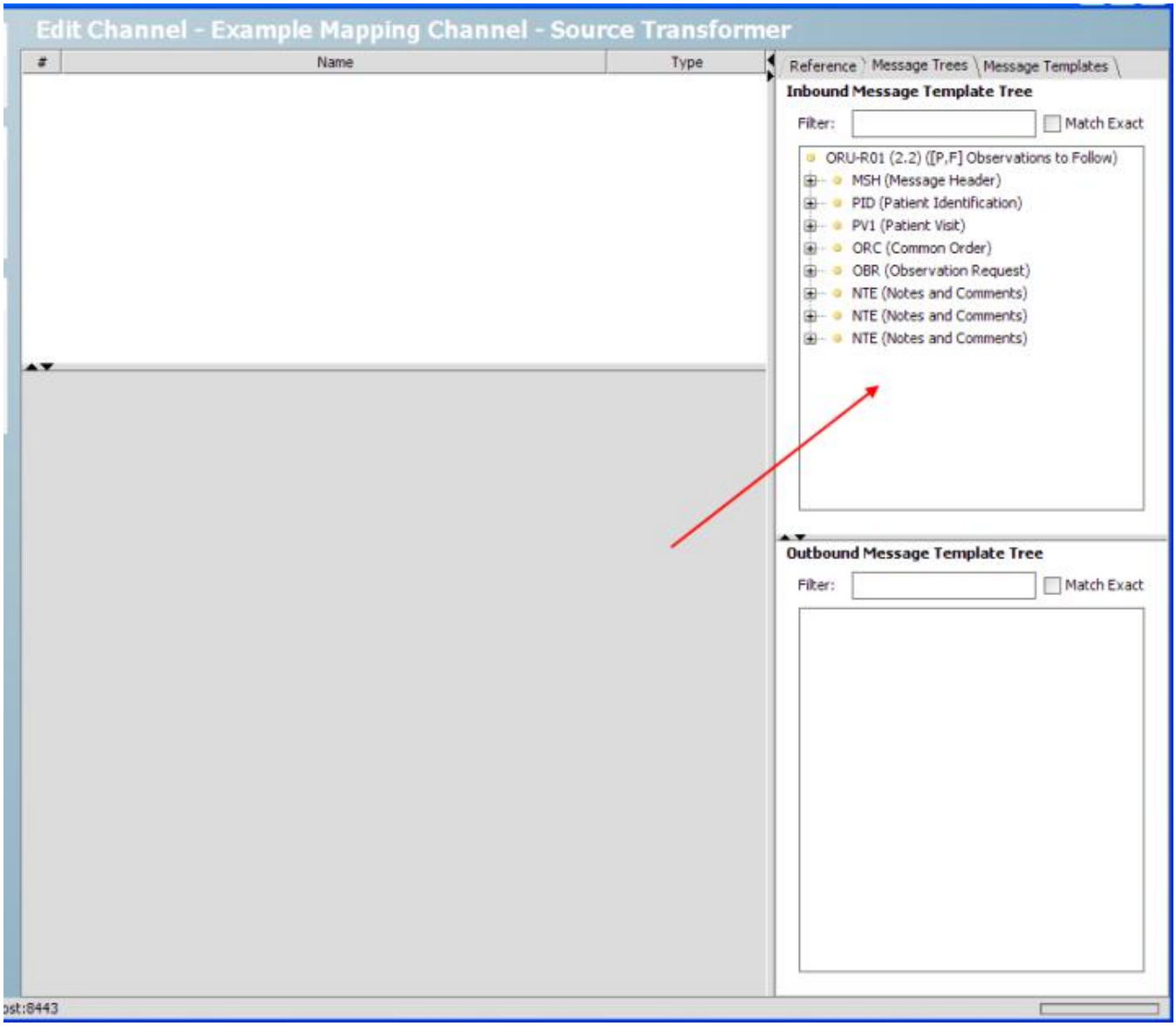
Step 1: Create a new channel, making sure that the Incoming Data field is set to HL7 v2.x.



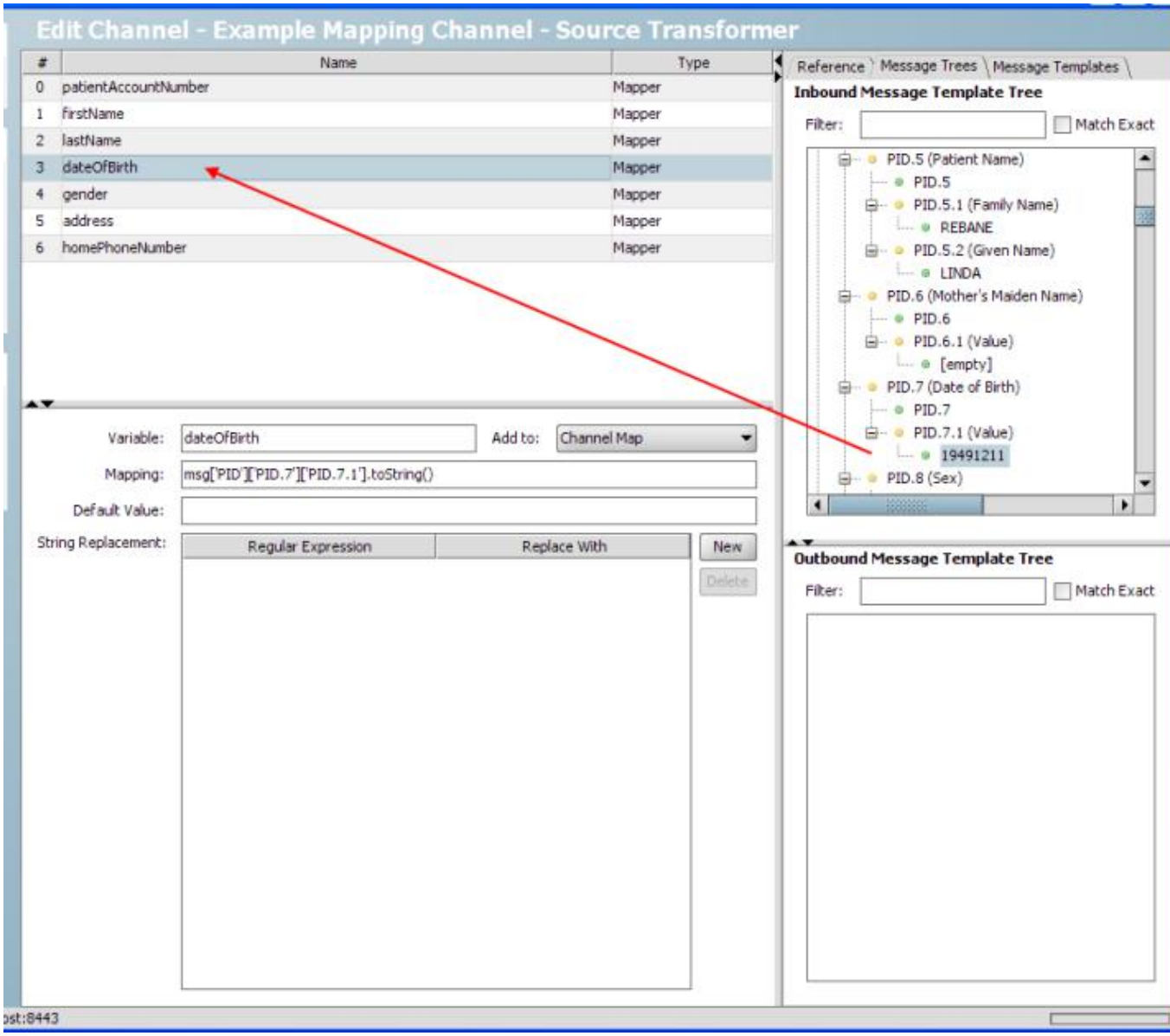
Step 2: Set the Source connector type to LLP Listener and configure any listener properties as needed. In this example, Gateway is listening for HL& messages on port 6661.



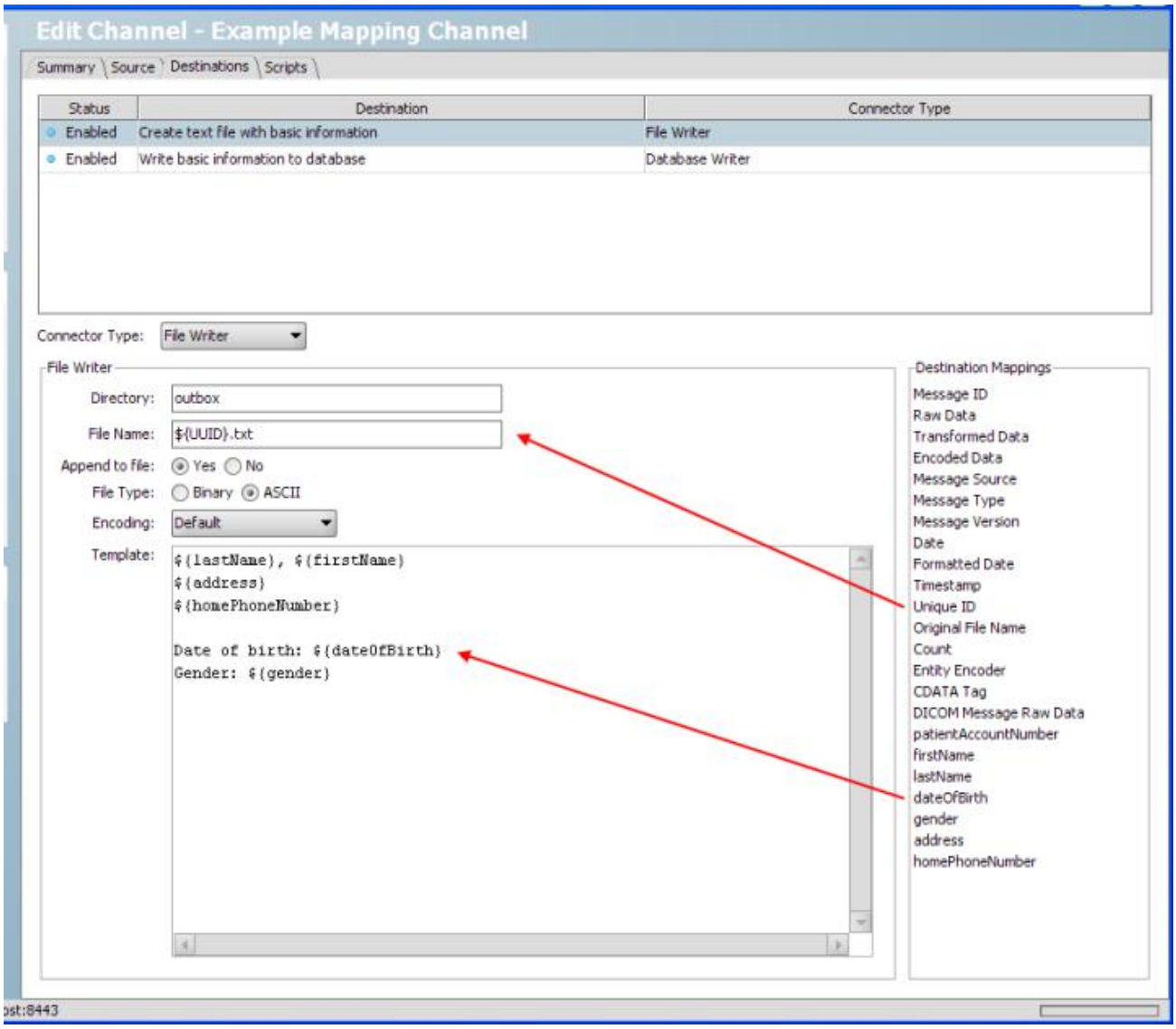
Step 3: Edit the Transformer for the Source connector and paste in a sample HL7 message into the Inbound Message Template field on the Message Templates tab.



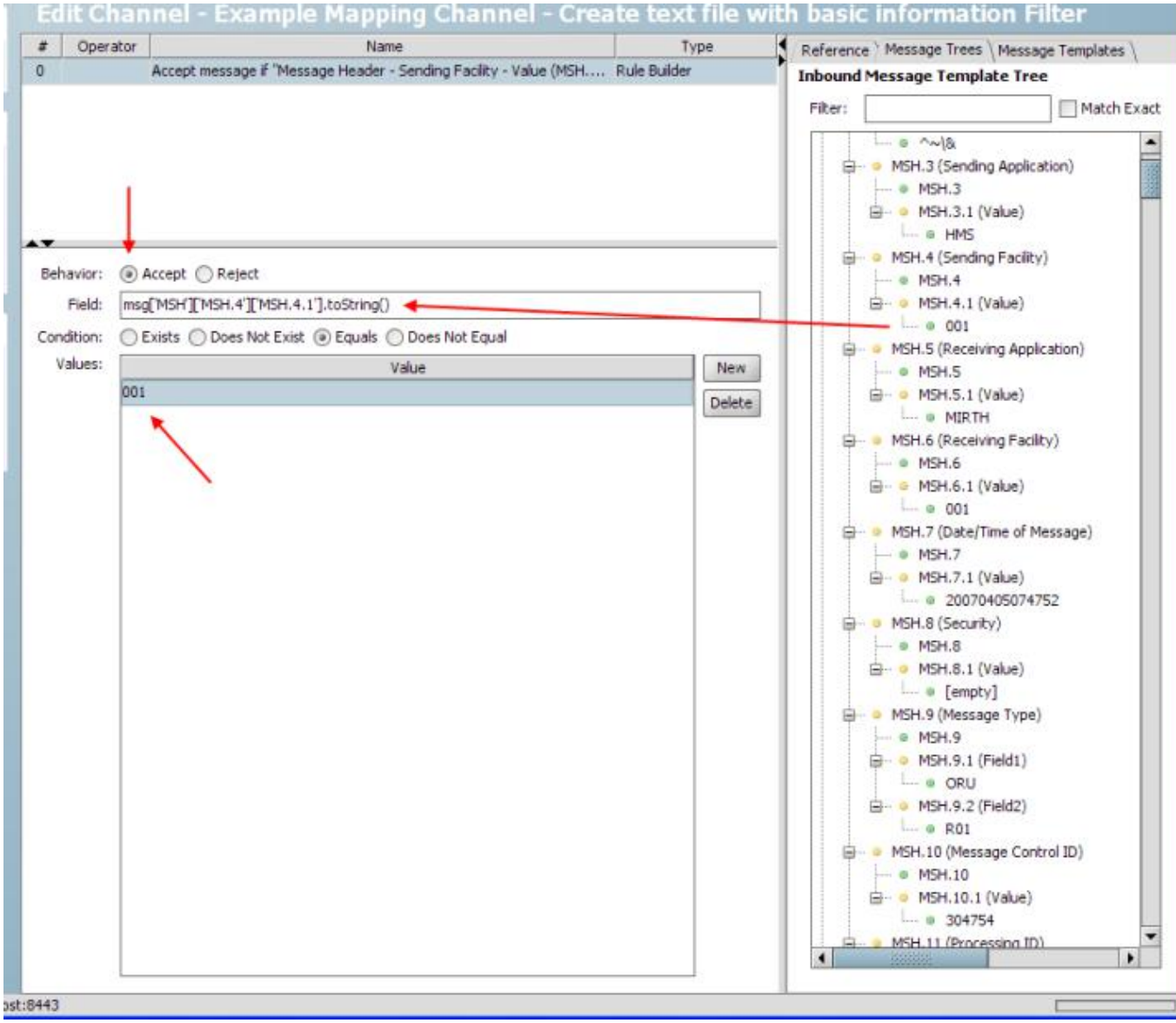
Step 4: Click on the Message Trees tab to view the tree generated for the Inbound Message Template. Right-clicking on any node in the tree displays a context menu to expand and collapse nodes in the tree.



Step 5: Drag nodes from the tree to the Transformer area to create new Mapping steps. In this example, seven mapping steps were created to extract patient information. Since these steps are in the transformer for the Source connector, they will be available to all destinations for this channel.



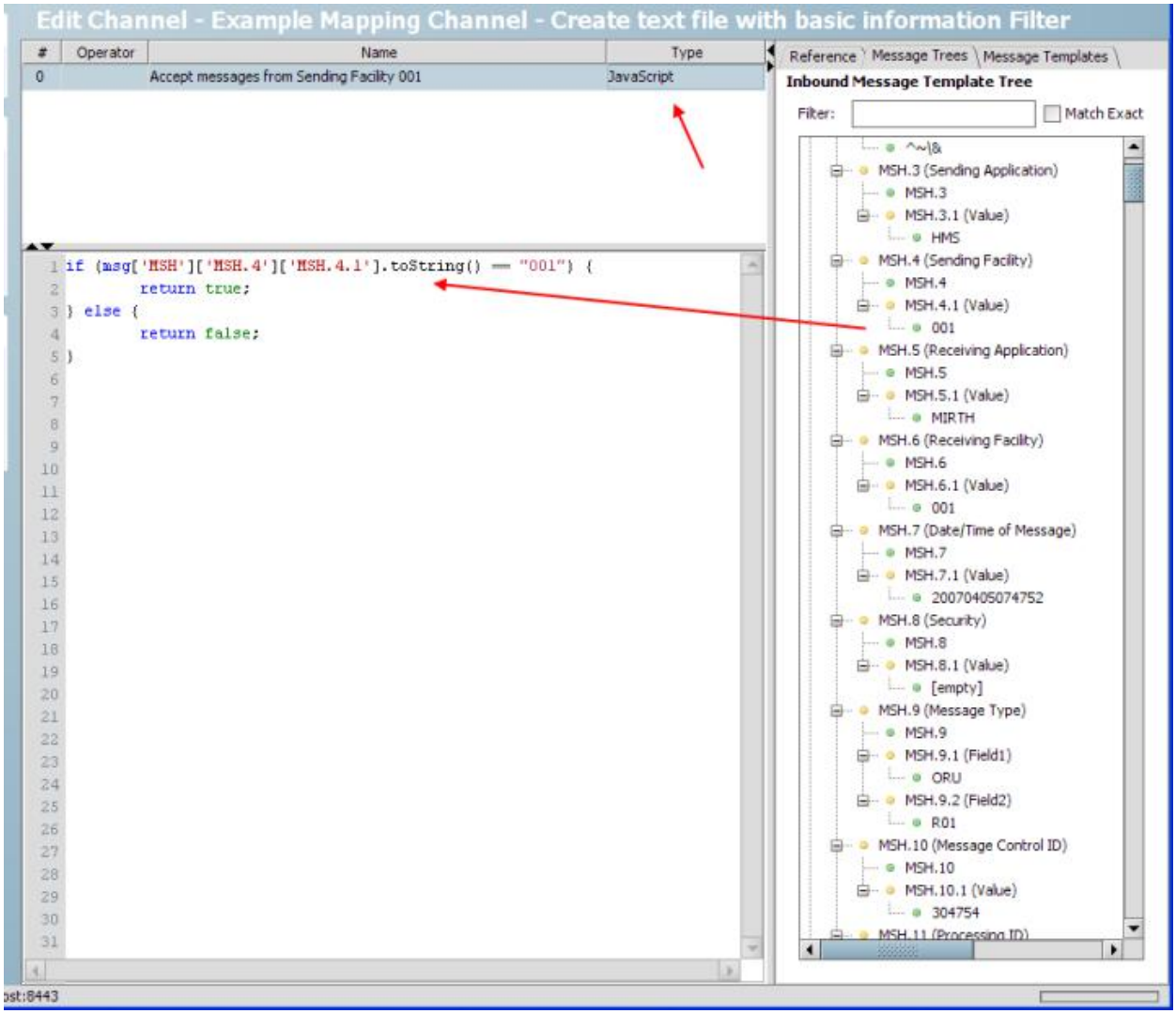
Step 6: In the Destinations area, create a new File Writer destination to generate a text file with the mapping data. Notice that the File Name field is set to a generated unique ID using the variable available in the Destination Mappings section on the right. The content of the text file as specified in the Template field also contains mappings, which were created in the transformer. When the channel is deployed and a message is received, the variables in the template will be replaced with the actual values from the message.

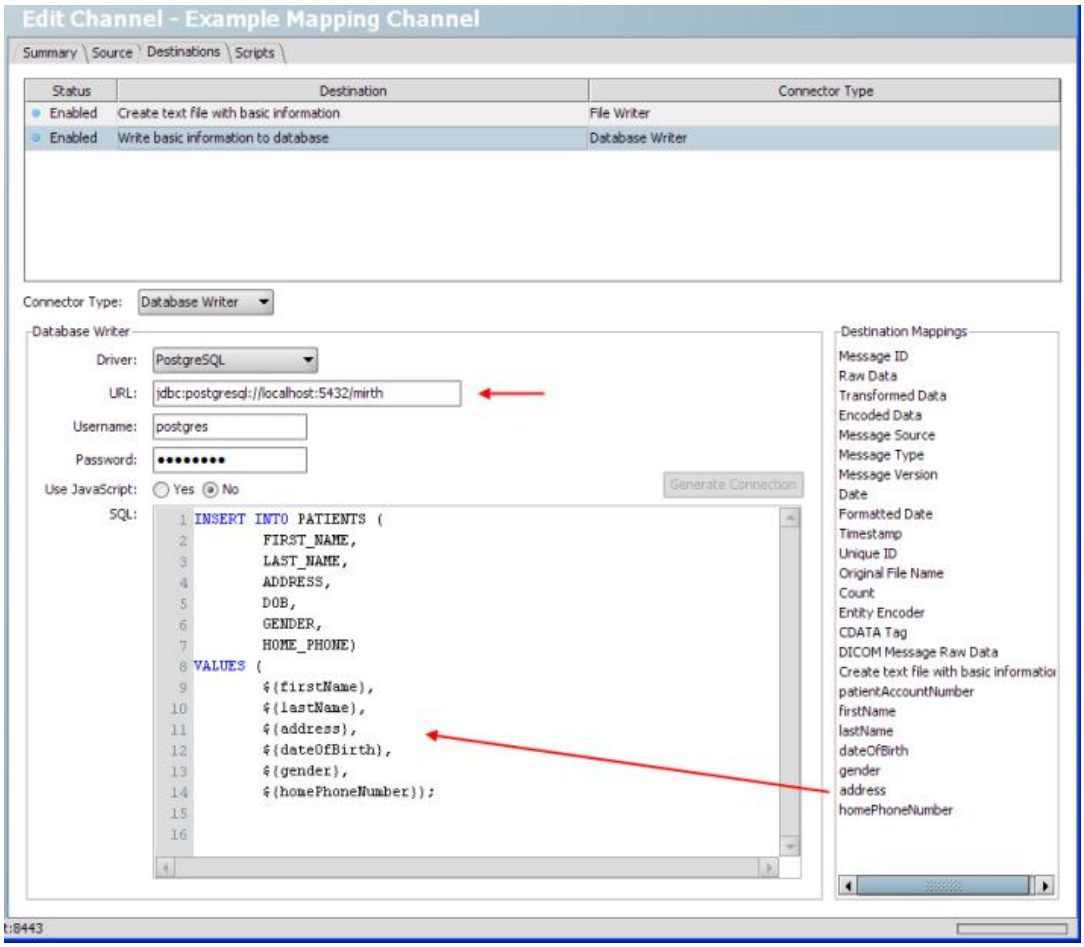


Step 7: Edit the Filter for the File Writer destination and create a new rule of type Rule Builder that only accepts messages with a sending facility of "001". The Field can be specified by dragging and dropping a node from the Inbound Message Template Tree if an Inbound Message Template has been specified.

When the channel is deployed and a message is received, a message will be sent to this destination only if the filter accepts the message.

The same rule can also be written in a JavaScript rule type. The rule can contain arbitrary JavaScript, but must return either TRUE to accept the message or FALSE. Multiple rules are joined using the AND operator by default, and can be changed to OR by modifying the Operator drop-down field. As with the Rule Builder step, nodes can be dragged and dropped into the JavaScript field from the Inbound Message Template tree.


Edit Channel - Example Mapping Channel - Create text file with basic information Filter
Operator Name Type
0 Accept messages from Sending Facility 001 JavaScript
if (msg['MSH']['MSH.4']['MSH.4.1'].toString() == "001") {
return true;
} else {
return false;
}
Reference Message Trees Message Templates
Inbound Message Template Tree
Filter:
Match Exact
MSH.3 (Sending Application)
MSH.3
MSH.3.1 (Value)
HMS
MSH.4 (Sending Facility)
MSH.4
MSH.4.1 (Value)
001
MSH.5 (Receiving Application)
MSH.5
MSH.5.1 (Value)
MIRTH
MSH.6 (Receiving Facility)
MSH.6
MSH.6.1 (Value)
001
MSH.7 (Date/Time of Message)
MSH.7
MSH.7.1 (Value)
20070405074752
MSH.8 (Security)
MSH.8
MSH.8.1 (Value)
[empty]
MSH.9 (Message Type)
MSH.9
MSH.9.1 (Field1)
ORU
MSH.9.2 (Field2)
R01
MSH.10 (Message Control ID)
MSH.10
MSH.10.1 (Value)
304754



Step 8: Create a new Database Writer destination to insert the extract message data into a PostgreSQL database instance. Notice that the URL entered for the database is a JDBC database connection string. Like the Template field in the File Writer, use any of the available mappings to form the INSERT statement.

3.3 使用 Apache HttpClient to HTTP GET 一个资源并直接写到文件

这是一个用 javascript 脚本写的转换步骤，创建通过 HTTP GET （使用 Apache HttpClient v4.0.1）获取文件的例子。

它说明如何：

- 使用 mportPackage 导入需要的包，不需要在调用每个对象类时使用全路径名称。
- 利用 Entity 的 HttpClient 4.0 函数把响应直接流入目标文件。

为此，你需先下载 HttpClient jar 文件及相关支持文件，放入医易通 \lib\custom 文件夹。 (See http://hc.apache.org/downloads.cgi)，现在3.0的版本已经不需要；

```
importPackage(Packages.org.apache.http.client);
importPackage(Packages.org.apache.http.client.methods);
importPackage(Packages.org.apache.http.impl.client);
```

```
varhttpclient =newDefaultHttpClient();

varhttpget =new HttpGet('http://myserver/mypath');
varresponse = httpclient.execute(httpget);
varentity = response.getEntity();
if(entity !=null) {
    var fos =new java.io.FileOutputStream('c:\\temp\\myfile.ext');
    entity.writeTo(fos);
    fos.close();
}
```

3.4 访问 HTTPS 服务的例子

由于涉及到证书的问题，所以，实现一个扩展类来提供服务端的证书储存库信息，并创建一个 HttpClient 对象来完成 调用：

```
package com.woohe.extension;
import java.io.File;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;

import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.ssl.SSLContexts;
import org.apache.http.util.EntityUtils;

public class ClientCustomSSL {
    public static CloseableHttpClient makeHttpClient(String store, String pwd) throws
Exception {
        // Trust own CA and all self-signed certs
        SSLContext sslcontext = SSLContexts.custom().loadTrustMaterial(new
File(store), pwd.toCharArray(), new TrustSelfSignedStrategy()).build();
        // Allow TLSv1 protocol only
        SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslcontext,
        new String[] { "TLSv1.1" }, null,
        new HostnameVerifier() {
```

```
            public boolean verify(String hostname, SSLSession session) {
                return true;
            }
        });
        // setConnectionTimeToLive(10, TimeUnit.MINUTES).
        return HttpClients.custom().setSSLSocketFactory(sslsf).build();

    }
//java 测试代码
    public final static void main(String[] args) throws Exception {
        try {
            CloseableHttpClient httpclient =
                    makeHttpClient("d:/opt/jssecacerts", "changeit");
            HttpPost post = new
HttpPost("https://localhost:8443/api/users/_login?username=admin&password=admin");
            post.setHeader("content-type", "application/x-www-form-urlencoded");
            CloseableHttpResponse response = httpclient.execute(post);

            HttpEntity entity = response.getEntity();
            System.out.println(response.getStatusLine());
            System.out.println(EntityUtils.toString(entity));
            response.close();

            HttpGet httpget = new HttpGet("https://localhost:8443/api/users");
            httpget.setHeader("content-type", "application/xml");
            response = httpclient.execute(httpget);
            entity = response.getEntity();
            System.out.println(response.getStatusLine());
            System.out.println(EntityUtils.toString(entity));
            response.close();
            httpclient.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

通道例子：

```
<channel version="3.4.0">
  <id>3ffc334a-7211-4e05-8817-753416ebb801</id>
  <nextMetaDataId>4</nextMetaDataId>
  <name>ssl 测试</name>
  <description> 例子： 使用定制扩展库，调用 https 服务。</description>
  <enabled>true</enabled>
```

```
  <lastModified>
    <time>1453195520731</time>
    <timezone>Asia/Shanghai</timezone>
  </lastModified>
  <revision>10</revision>
  <sourceConnector version="3.4.0">
    <metaDataId>0</metaDataId>
    <name>sourceConnector</name>
    <properties      class="com.woohe.connect.connectors.http.HttpReceiverProperties"
version="3.4.0">
      <pluginProperties/>
      <listenerConnectorProperties version="3.4.0">
        <host>0.0.0.0</host>
        <port>802</port>
      </listenerConnectorProperties>
      <sourceConnectorProperties version="3.4.0">
        <responseVariable>d2</responseVariable>
        <respondAfterProcessing>true</respondAfterProcessing>
        <processBatch>false</processBatch>
        <firstResponse>false</firstResponse>
        <resourceIds class="singleton-set">
          <string>Default Resource</string>
        </resourceIds>
      </sourceConnectorProperties>
      <isHttps>false</isHttps>
      <xmlBody>true</xmlBody>
      <parseMultipart>true</parseMultipart>
      <includeMetadata>false</includeMetadata>

<binaryMimeTypes>application/.*(?&amp;lt;!json|xml)$|image/.*|video/.*|audio/.*</binary
MimeTypes>
      <binaryMimeTypesRegex>true</binaryMimeTypesRegex>
      <responseContentType>text/plain</responseContentType>
      <responseDataTypeBinary>false</responseDataTypeBinary>
      <responseStatusCode></responseStatusCode>
      <responseHeaders class="linked-hash-map"/>
      <charset>UTF-8</charset>
      <contextPath>ssl</contextPath>
      <timeout>0</timeout>
      <staticResources/>
    </properties>
    <transformer version="3.4.0">
      <steps>
        <step>
```

```
          <sequenceNumber>0</sequenceNumber>
          <name>New Step</name>
          <script>var
act=sourceMap.get(&apos;parameters&apos;).getParameter(&apos;action&apos;);
destinationSet.removeAllExcept([act]);


logger.info(sourceMap.get(&apos;contextPath&apos;)+&quot;:&quot;+sourceMap.get(&apo
s;method&apos;)+&quot;:&quot;+
sourceMap.get(&apos;parameters&apos;).getParameter(&apos;action&apos;));</script>
          <type>JavaScript</type>
          <data>
            <entry>
              <string>Script</string>
              <string>var
act=sourceMap.get(&apos;parameters&apos;).getParameter(&apos;action&apos;);
destinationSet.removeAllExcept([act]);


logger.info(sourceMap.get(&apos;contextPath&apos;)+&quot;:&quot;+sourceMap.get(&apo
s;method&apos;)+&quot;:&quot;+
sourceMap.get(&apos;parameters&apos;).getParameter(&apos;action&apos;));</string>
            </entry>
          </data>
        </step>
      </steps>
      <inboundTemplate encoding="base64"></inboundTemplate>
      <outboundTemplate encoding="base64"></outboundTemplate>
      <inboundDataType>XML</inboundDataType>
      <outboundDataType>JSON</outboundDataType>
      <inboundProperties
class="com.woohe.connect.plugins.datatypes.xml.XMLDataTypeProperties"
version="3.4.0">
        <serializationProperties
class="com.woohe.connect.plugins.datatypes.xml.XMLSerializationProperties"
version="3.4.0">
          <stripNamespaces>false</stripNamespaces>
        </serializationProperties>
        <batchProperties
class="com.woohe.connect.plugins.datatypes.xml.XMLBatchProperties" version="3.4.0">
          <splitType>Element_Name</splitType>
          <elementName></elementName>
          <level>1</level>
          <query></query>
```

```
          <batchScript></batchScript>
        </batchProperties>
      </inboundProperties>
      <outboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
        <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
          <batchScript></batchScript>
        </batchProperties>
      </outboundProperties>
    </transformer>
    <filter version="3.4.0">
      <rules/>
    </filter>
    <transportName>HTTP Listener</transportName>
    <mode>SOURCE</mode>
    <enabled>true</enabled>
    <waitForPrevious>true</waitForPrevious>
  </sourceConnector>
  <destinationConnectors>
    <connector version="3.4.0">
      <metaDataId>2</metaDataId>
      <name>getstate</name>
      <properties
class="com.woohe.connect.connectors.js.JavaScriptDispatcherProperties"
version="3.4.0">
        <pluginProperties/>
        <destinationConnectorProperties version="3.4.0">
          <queueEnabled>false</queueEnabled>
          <sendFirst>false</sendFirst>
          <retryIntervalMillis>10000</retryIntervalMillis>
          <regenerateTemplate>false</regenerateTemplate>
          <retryCount>0</retryCount>
          <rotate>false</rotate>
          <includeFilterTransformer>false</includeFilterTransformer>
          <threadCount>1</threadCount>
          <threadAssignmentVariable></threadAssignmentVariable>
          <validateResponse>false</validateResponse>
          <resourceIds class="singleton-set">
            <string>Default Resource</string>
          </resourceIds>
        </destinationConnectorProperties>
```

```
       <script>//importPackage(Packages.org.apache.http.client);
//importPackage(Packages.org.apache.http.client.methods);
//importPackage(Packages.org.apache.http.impl.client);


    var h= globalChannelMap.get(&amp;quot;htc&amp;quot;);
    var                                                                    pp=new
org.apache.http.client.methods.HttpGet(&amp;quot;https://localhost:8443/api/users&amp;quot;)
;
     r=h.execute(pp);
     var ret=org.apache.http.util.EntityUtils.toString(r.getEntity());
     r.close();
    logger.info(ret);
    return ret;</script>
     </properties>
     <transformer version="3.4.0">
       <steps/>
       <inboundDataType>JSON</inboundDataType>
       <outboundDataType>JSON</outboundDataType>
       <inboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
         <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </inboundProperties>
       <outboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
         <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </outboundProperties>
     </transformer>
     <responseTransformer version="3.4.0">
       <steps/>
       <inboundTemplate encoding="base64"></inboundTemplate>
       <outboundTemplate encoding="base64"></outboundTemplate>
       <inboundDataType>JSON</inboundDataType>
       <outboundDataType>JSON</outboundDataType>
```

```
       <inboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
         <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </inboundProperties>
       <outboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
         <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </outboundProperties>
     </responseTransformer>
     <filter version="3.4.0">
       <rules/>
     </filter>
     <transportName>JavaScript Writer</transportName>
     <mode>DESTINATION</mode>
     <enabled>true</enabled>
     <waitForPrevious>true</waitForPrevious>
   </connector>
   <connector version="3.4.0">
     <metaDataId>3</metaDataId>
     <name>default</name>
     <properties
class="com.woohe.connect.connectors.js.JavaScriptDispatcherProperties"
version="3.4.0">
       <pluginProperties/>
       <destinationConnectorProperties version="3.4.0">
         <queueEnabled>false</queueEnabled>
         <sendFirst>false</sendFirst>
         <retryIntervalMillis>10000</retryIntervalMillis>
         <regenerateTemplate>false</regenerateTemplate>
         <retryCount>0</retryCount>
         <rotate>false</rotate>
         <includeFilterTransformer>false</includeFilterTransformer>
         <threadCount>1</threadCount>
         <threadAssignmentVariable></threadAssignmentVariable>
```

```
          <validateResponse>false</validateResponse>
          <resourceIds class="singleton-set">
            <string>Default Resource</string>
          </resourceIds>
        </destinationConnectorProperties>
        <script>


    return &amp;quot;请根据需要在请求路径上提供 action 和参数、内容，每个 action 对应服务端请求的路径，并且选择不同的目标插头执行 ssl 请求&amp;quot;;</script>
      </properties>
      <transformer version="3.4.0">
        <steps/>
        <inboundDataType>JSON</inboundDataType>
        <outboundDataType>JSON</outboundDataType>
        <inboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
          <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
            <batchScript></batchScript>
          </batchProperties>
        </inboundProperties>
        <outboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
          <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
            <batchScript></batchScript>
          </batchProperties>
        </outboundProperties>
      </transformer>
      <responseTransformer version="3.4.0">
        <steps/>
        <inboundTemplate encoding="base64"></inboundTemplate>
        <outboundTemplate encoding="base64"></outboundTemplate>
        <inboundDataType>JSON</inboundDataType>
        <outboundDataType>JSON</outboundDataType>
        <inboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
          <batchProperties
```

```
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </inboundProperties>
       <outboundProperties
class="com.woohe.connect.plugins.datatypes.json.JSONDataTypeProperties"
version="3.4.0">
         <batchProperties
class="com.woohe.connect.plugins.datatypes.json.JSONBatchProperties"
version="3.4.0">
           <batchScript></batchScript>
         </batchProperties>
       </outboundProperties>
     </responseTransformer>
     <filter version="3.4.0">
       <rules/>
     </filter>
     <transportName>JavaScript Writer</transportName>
     <mode>DESTINATION</mode>
     <enabled>true</enabled>
     <waitForPrevious>true</waitForPrevious>
   </connector>
 </destinationConnectors>
 <preprocessingScript>// Modify the message variable below to pre process data
return message;</preprocessingScript>
 <postprocessingScript>// This script executes once after a message has been processed
// Responses returned from here will be stored as &quot;Postprocessor&quot; in the
response map
return;</postprocessingScript>
 <deployScript>// This script executes once when the channel is deployed
// You only have access to the globalMap and globalChannelMap here to persist data
//java.lang.System.setProperty(&quot;javax.net.ssl.keyStore&quot;,
&quot;d:\\opt\\mirth5\\server\\ca\\2016\\yytstore&quot;);
//  java.lang.System.setProperty(&quot;javax.net.ssl.keyStorePassword&quot;,
&quot;wooheconnect&quot;);
                                        var                                    h=
com.woohe.extension.ClientCustomSSL.makeHttpClient(&quot;d:/opt/jssecacerts&quot;,
&quot;changeit&quot;);
    var                      pp                     =                      new
org.apache.http.client.methods.HttpPost(&quot;https://localhost:8443/api/users/_log
in?username=admin&amp;password=admin&quot;);
    pp.setHeader(&quot;content-type&quot;,
&quot;application/x-www-form-urlencoded&quot;);
```

```
    h.execute(pp);
    globalChannelMap.put(&amp;quot;htc&amp;quot;,h);
return;</deployScript>
  <undeployScript>// This script executes once when the channel is undeployed
// You only have access to the globalMap and globalChannelMap here to persist data
return;</undeployScript>
  <properties version="3.4.0">
    <clearGlobalChannelMap>true</clearGlobalChannelMap>
    <messageStorageMode>DEVELOPMENT</messageStorageMode>
    <encryptData>false</encryptData>
    <removeContentOnCompletion>false</removeContentOnCompletion>
    <removeOnlyFilteredOnCompletion>false</removeOnlyFilteredOnCompletion>
    <removeAttachmentsOnCompletion>false</removeAttachmentsOnCompletion>
    <initialState>STARTED</initialState>
    <storeAttachments>false</storeAttachments>
    <tags class="linked-hash-set"/>
    <metaDataColumns>
      <metaDataColumn>
        <name>SOURCE</name>
        <type>STRING</type>
        <mappingName>woohe_source</mappingName>
      </metaDataColumn>
      <metaDataColumn>
        <name>TYPE</name>
        <type>STRING</type>
        <mappingName>woohe_type</mappingName>
      </metaDataColumn>
    </metaDataColumns>
    <attachmentProperties>
      <type>None</type>
      <properties/>
    </attachmentProperties>
    <archiveEnabled>true</archiveEnabled>
    <resourceIds class="linked-hash-set">
      <string>Default Resource</string>
    </resourceIds>
  </properties>
  <codeTemplateLibraries/>
</channel>
```

例子通道里面的主要代码说明：

通道的发布脚本（deploy)：创建 httpclient 对象 htc，并执行向服务端的登录请求。准备好可以重复执行请求的客户端对象。

```
 var h= com.woohe.extension.ClientCustomSSL.makeHttpClient("d:/opt/jssecacerts",
"changeit");//调整你的证书库文件路径和密码。
    //执行登录请求。（以登录医易通客户端 api 接口为例）。
    var pp = new org.apache.http.client.methods.HttpPost(
        "https://localhost:8443/api/users/_login?username=admin&password=admin");
    pp.setHeader("content-type", "application/x-www-form-urlencoded");
    h.execute(pp);
    globalChannelMap.put("htc",h);
```

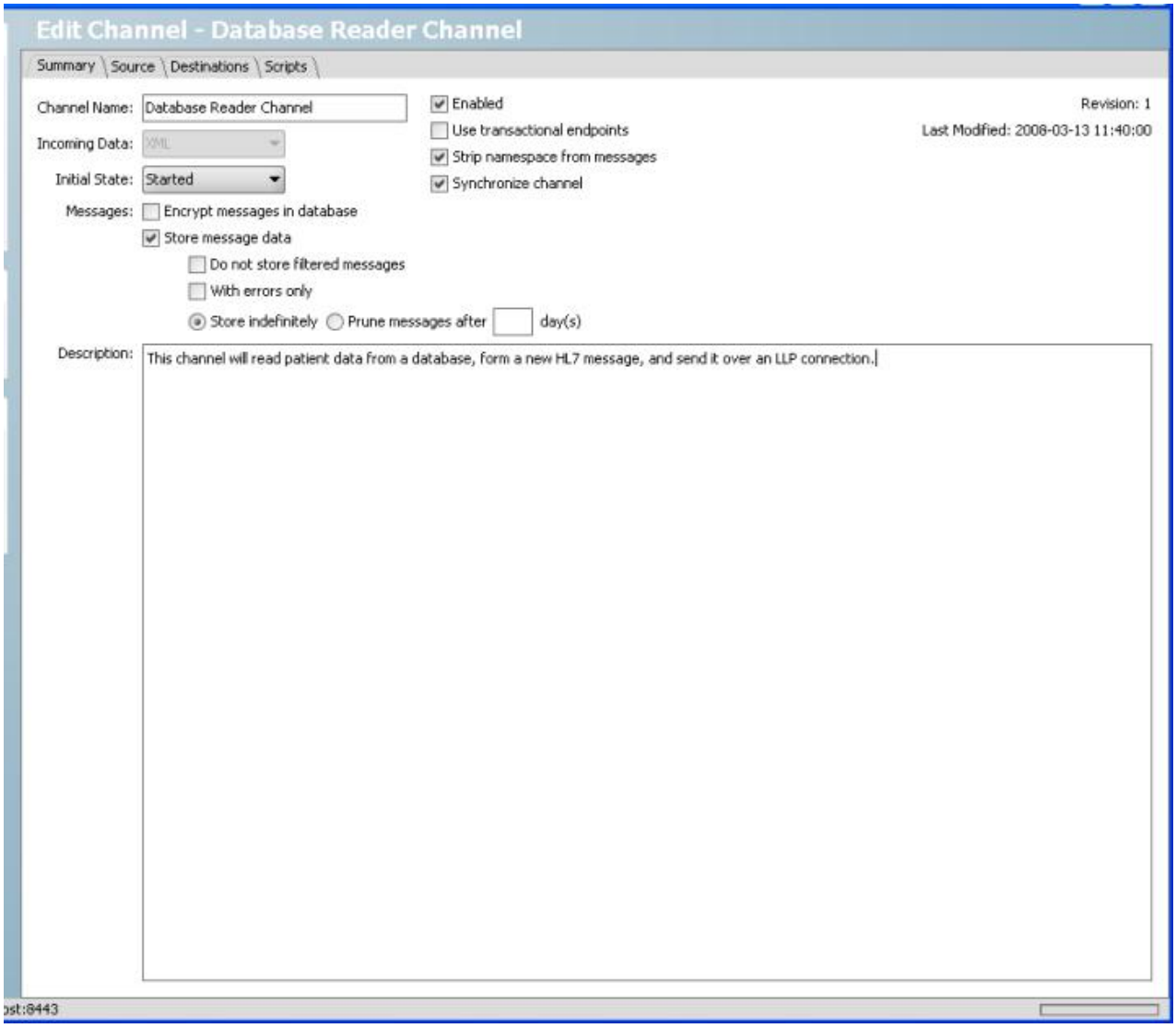
在 JSwriter 插头里执行请求的 js 代码例子：

```
    importPackage(Packages.org.apache.http.client);
    importPackage(Packages.org.apache.http.client.methods);
    importPackage(Packages.org.apache.http.util);

    var h= globalChannelMap.get("htc");//获取通道映射中保存的 httpclient 对象。
    var pp=new HttpGet("https://localhost:8443/api/users");
    r=h.execute(pp);
    System.out.println(response.getStatusLine());//打印请求响应状态行
    var ret=EntityUtils.toString(r.getEntity());
    r.close();
    logger.info(ret);
    return ret;
```

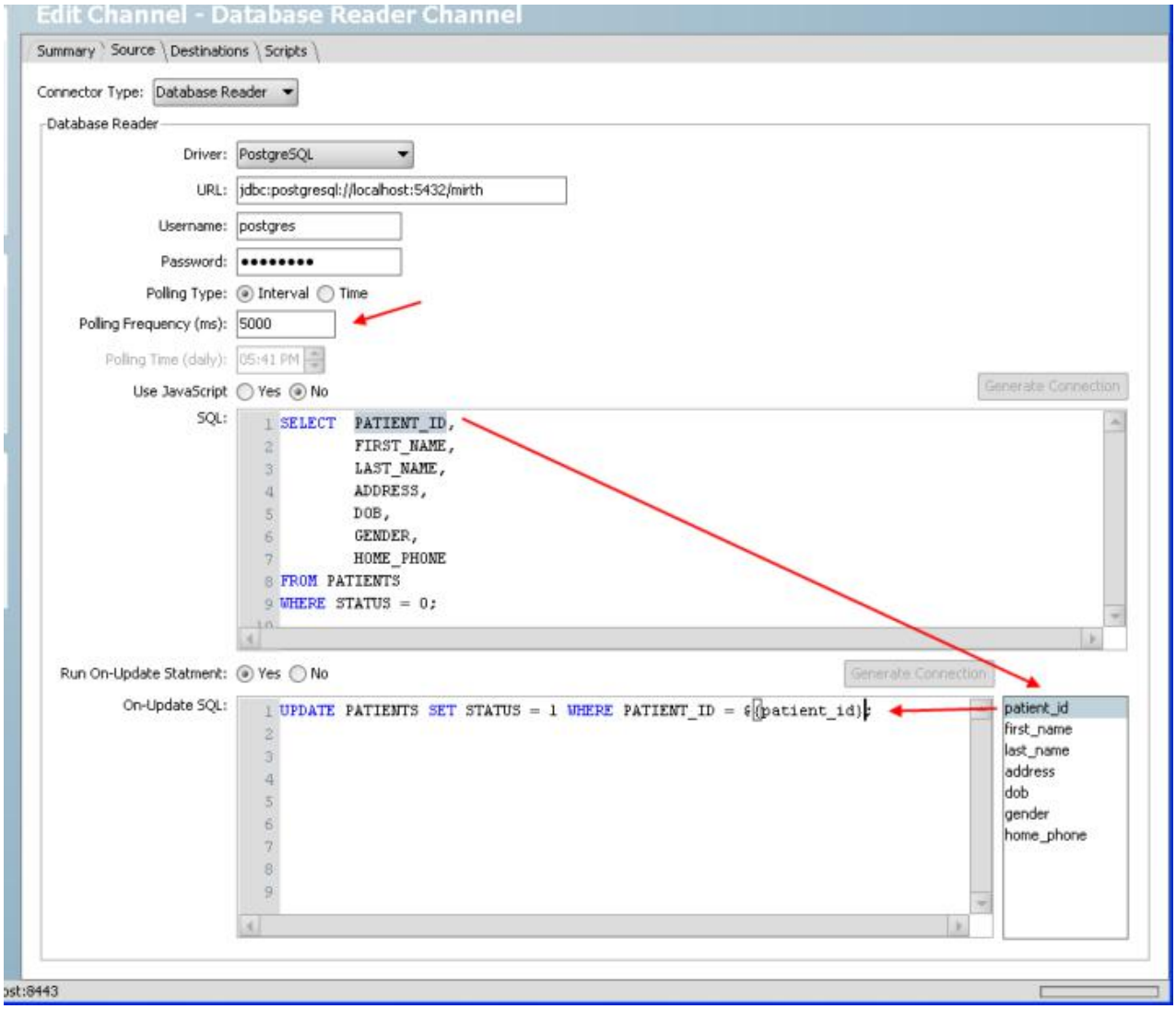
3.5 消息构建器

This example will show how to configure a channel to read data from a database, form an HL7 v2.x message using the data, and send it over an LLP connection.
This example will demonstrate:

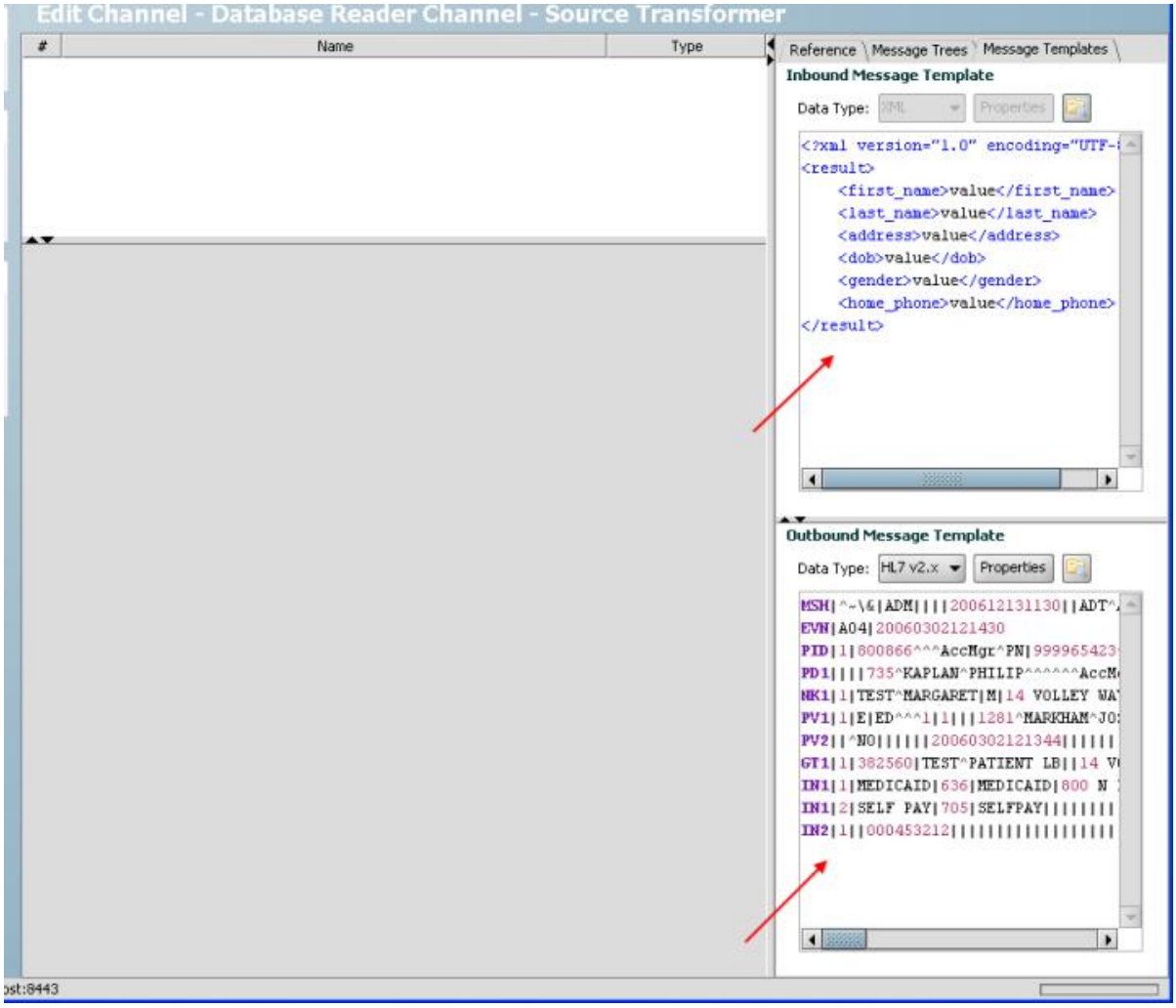
- Using the Database Reader
- Generating an HL7 v2.x message from a template



Step 1: Create a new channel.

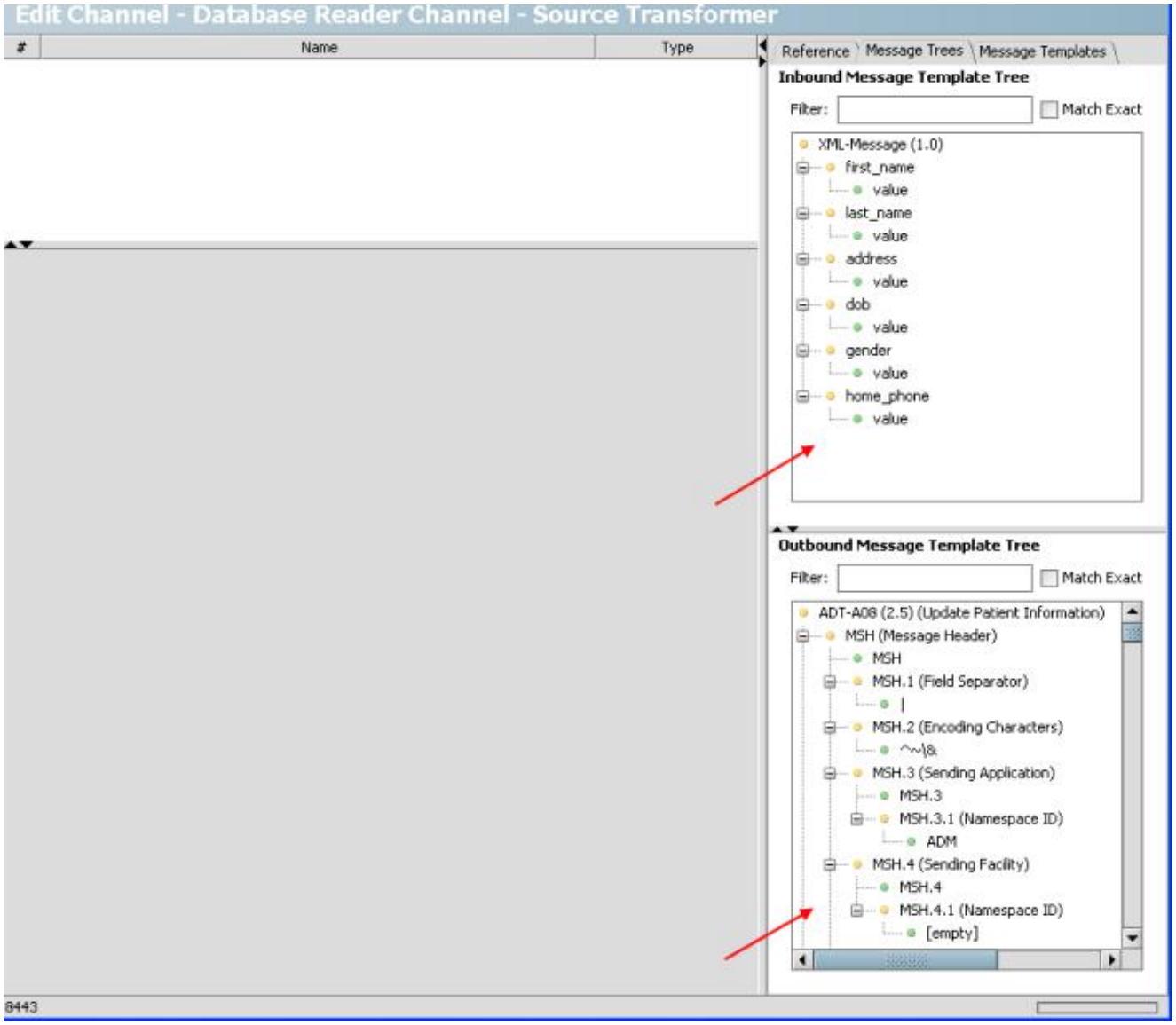


Step 2: Set to Source connector type to Database Reader and enter the connection information. In this example, the connector is set to poll the database every 5 seconds and execute the specified **SELECT** query. Columns entered in the **FROM** clause are automatically added as available variables. Note that the On-Update SQL statement that is executed immediately after the query can make use of variables returned from the initial query.

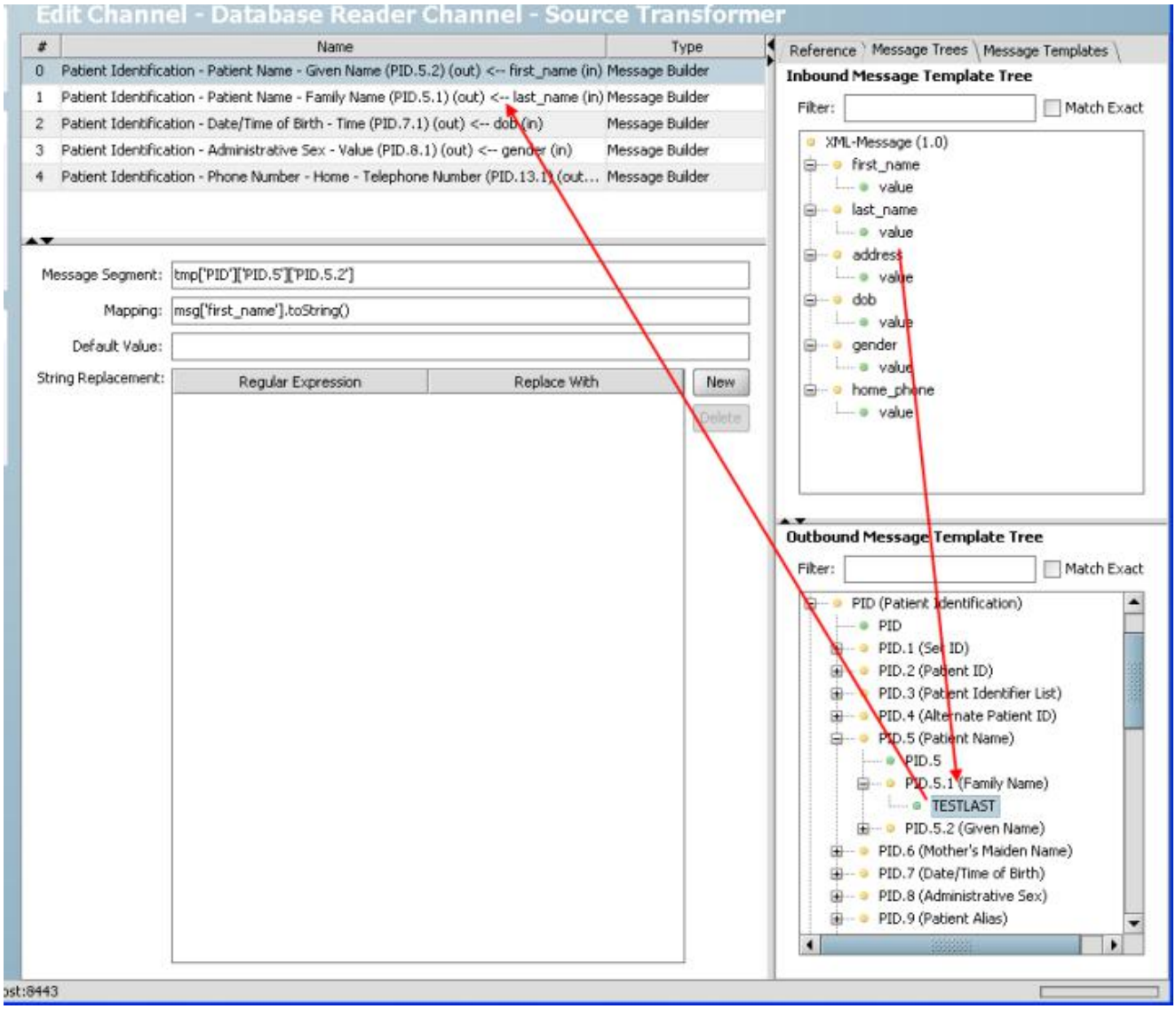


Step 3: Edit the transformer for the Source connector and click on the Message Templates tab to paste in an example Outbound Message Template, making sure to set the Data Type to HL7 v2.x. This template will be used as the base for new outbound HL7 messages generated by the channel.

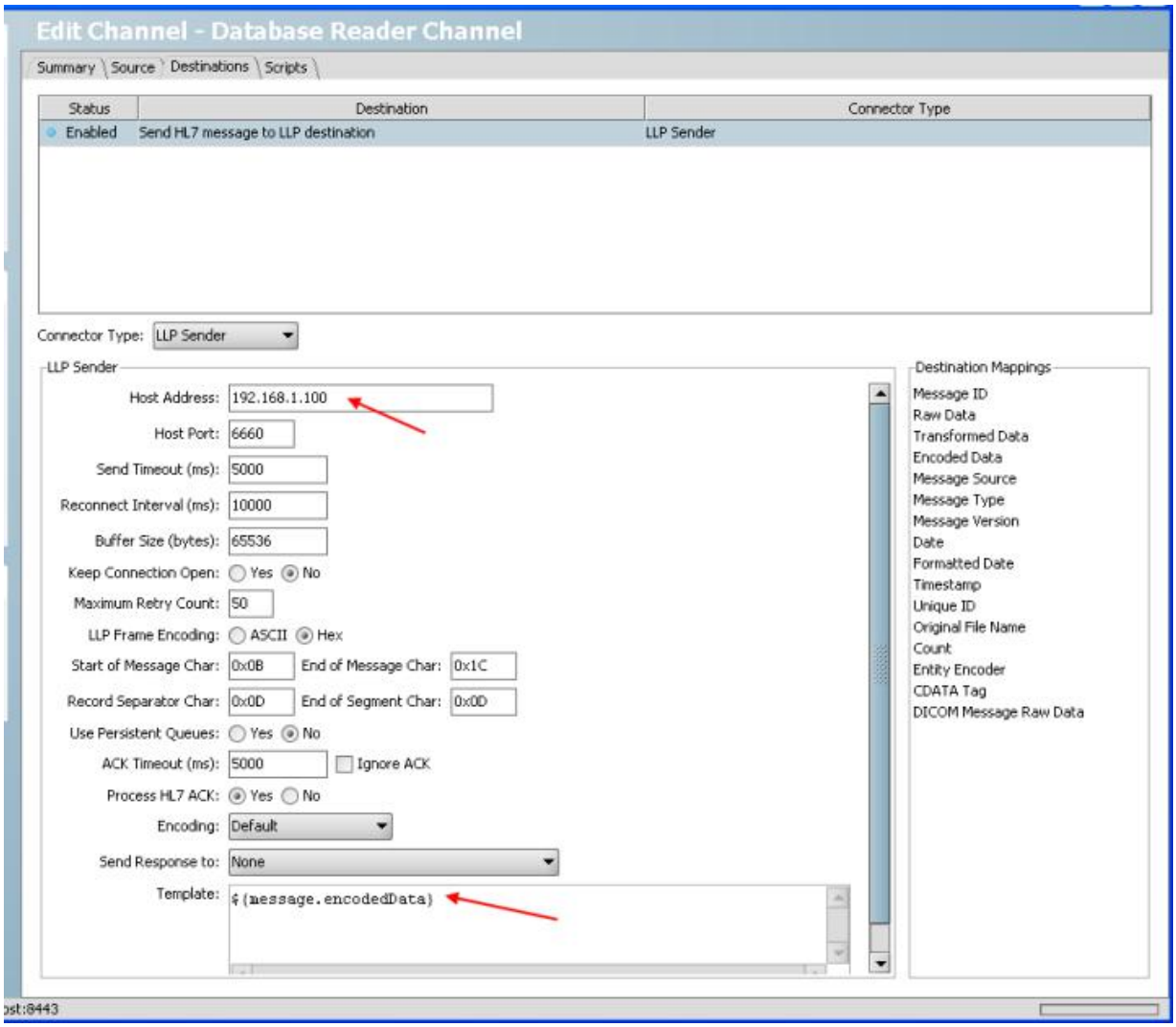
The Inbound Message Template is an example of what an inbound message resulting from the database query will look like. This shows how the XML has been generated using the columns selected.



Step 4: Click on the Message Trees tab to view the trees for both the Inbound and Outbound message templates.



Step 5: In order to map data from the database to the message template, click on a node in the Inbound tree and drag-and-drop it over the node in the Outbound tree which should be replaced. This example shows replacing the "PID.5.1 (Family Name)" field in the message template with the "last_name" variable from the database. The auto-generated name of the transformer step reflects this mapping.



Step 6: Add a new destination connector of type LLP Sender. The Template is the data that will be sent over the LLP connection to the host address and by default is set to the Encoded Data, which in this example is an HL7 v2.x message.

3.6 例子：查询数据库 过滤器

```
//test existence of PatientId in database
  //Simply use the value in the class= specification
var driver ="net.sourceforge.jtds.jdbc.Driver";
var address ="jdbc:jtds:sqlserver://srv.test.com:1433/yourdb";
var username ="demo";
var password ="demo";
//DatabaseConnection
  var dbConn = DatabaseConnectionFactory.createDatabaseConnection(driver, address, username,
password);
//the query
var patId10 = msg['PID']['PID.2']['CK.1'];
var expression ="SELECT COUNT(*) FROM Patient WHERE PatId10 = '"+ patId10 +"';";
//CachedRowSetImpl
var result = dbConn.executeCachedQuery(expression);
```

```
//I am not sure if the following still works so you may want
//to just try just the result variable for a single result

//go to the first result
result.next();
//get the value from the first column as an integer
var iCount = result.getInt(1);

//cleanup
//If you can get an EcmaError running gateway
//saying "Cannot find function close" comment out result.close()
//next line. This was needed using executeUpdateQuery

result.close();
dbConn.close();

//if the PatientId doesn't exists we may do the insert
return(iCount ==0);
```

3.7 例子：用数据库值映射到 HL7 消息字段

用数据库值替换 HL7 消息字段

```
//assign values to variable.
//Variables are of type Mapper from HL7 message.
var drNum = $('STF1_1');
var drName = $('STF3_1');
var reason = $('MFI3_1');
//Connection String
var dbConn = DatabaseConnectionFactory.createDatabaseConnection
('net.sourceforge.jtds.jdbc.Driver','jdbc:jtds:sqlserver://ServerName:Port/dbaseName','UserName','Pass
word');
var Query ="SELECT TYPE_CD,NAME,INACT_IND FROM CODE_TABLE WHERE TYPE_CD
LIKE '%"+drNum+"%'";;
var result = dbConn.executeCachedQuery(Query);
while(result.next())
{
//Trim Example
//var hold = result.getString("TYPE_CD").trim();
//msg['MSH']['MSH.3']['MSH.3.1'] = hold;
//Parse through result and assign to variables.
//This assumes that only one record is returned and not more than one so there is no logic here
//("TYPE_CD") is a column name in the database and so are the other two
var typecd = result.getString("TYPE_CD").trim();
var name = result.getString("NAME").trim();
var inactind = result.getString("INACT_IND").trim();
//Assign variables to field. Be sure to use send ${message.encodedData} in your Destination Template
msg['STF']['STF.2']['STF.2.1'] = typecd;
msg['STF']['STF.3']['STF.3.1'] = name;
msg['MFI']['MFI.3']['MFI.3.1'] = inactind;
}
```

```
dbConn.close();
return result;
```

3.8 例子：映射多个 HL7 消息字段到数据库

该例子解释 如何处理当前消息中多个相同类型的消息段映射到数据库的问题。 例如,一个具有多个亲属的多个NK1段的ADT^A04消息 。

如果我们想要在集成交换引擎存储这些信息,转换器当然不能预测消息的同类型子段数量。当然您可以指定一个特定的最大值来处理,但本例子将教您能够映射每个子段(或任何其他部分)到您的数据库表中。

将以下所有代码放入一个过滤器：

```
//first thing you may want to do here is ensure that you are
// only dealing with messages that have the segments you are
// looking for. In this example, we will deal with ADT^A04 messages.

if(msg['MSH']['MSH.9']['CM_MSG.1'] == "ADT" && msg['MSH']['MSH.9']['CM_MSG.2'] == "A04")
{
//the driver will depend on your database. we will use MySQL
var driver = "com.mysql.jdbc.Driver";
var address = "jdbc:mysql://localhost/rawhl7";
var username = "user";
var password = "password";

var dbConn =
DatabaseConnectionFactory.createDatabaseConnection(driver,address,username,password);

//in my bridge table that is storing the kin for my patients, I need the patient ID to identify
my patient

var patientID = msg['PID']['PID.3']['CX.1'];

var i = 0;

//whenever you have more than one segment, yyt essentially creates an array of them. Which you
can access traditionally varname[index].

while(msg['NK1']['NK1.2']['XPN.2'][i] != null) {

var kinsFirstName= msg['NK1']['NK1.2']['XPN.2'][i];

var expression = "INSERT INTO kinTable values (DEFAULT, '" + patientID + "', '" + kinsFirstName
+ "');";

var result = dbConn.executeUpdate(expression);

i = i + 1;
}
//simply closes the connection to the DB
dbConn.close();
```

```
}

//finally you can return whatever you want to do with your filter.
return true;

// 有用的代码： 将 hl7 消息分解为多个段
```

```
tmp = <delimited/>;
var row;

for each (seg in msg.children()) {//msg: xml 化的消息
	var segName = seg.name().toString();

	switch(segName) {
		case 'MSH':
			if (row) {
				tmp.appendChild(row);
			}
			row = <row><internalID/><lastName/><firstName/><DOB/></row>;
			break;
		case 'PID':
			row.internalID = seg['PID.3']['PID.3.1'].toString();
			row.lastName = seg['PID.5']['PID.5.1'].toString();
			row.firstName = seg['PID.5']['PID.5.2'].toString();
			row.DOB = seg['PID.7']['PID.7.1'].toString();
	}
}

if (row)
	tmp.appendChild(row);
```

3.9 例子：映射、翻译、对码消息字段到数据库查询

可以把复杂的功能写到代码模板和全局脚本里。

This is a transformer step I use for one of my channels - populates OBR2.1 with accession numbers looked up from a Postgres table that is keyed off of several patient demographic values. (I removed a bunch of sanity checking to keep this example readable.) Shouldn't be hard to adjust to what you need...

```
var paramList = Packages.java.util.ArrayList();
var dbConn = DatabaseConnectionFactory.createDatabaseConnection ('org.postgresql.Driver',
'jdbcostgresql://172.17.18.16:5432/yytmlo', 'uname', 'passwd');
```

```
lname = msg['PID']['PID.5']['PID.5.1'].toString();
fname = msg['PID']['PID.5']['PID.5.2'].toString();
dob = msg['PID']['PID.7']['PID.7.1'].toString();
ssn = msg['PID']['PID.19']['PID.19.1'].toString();
dos = msg['OBR']['OBR.7']['OBR.7.1'].toString().substring(0,8);

var criteria = lname + "^" + fname + "^" + dob + "^" + ssn + "^" + dos;
paramList.add(criteria);
var Query = "SELECT info FROM InfoHold WHERE criteria=?";
var result = dbConn.executeCachedQuery(Query, paramList);

if (result.next()) {
acn = result.getString("info");

for each (obr in msg..OBR) {
obr['OBR.2']['OBR.2.1'] = acn;
}
} else {
channelMap.put("nomatchfound", 1);
}
```

3.10 例子：javascrip reader 读取数据库记录，组装 XML 消息

```
// 数据库连接
var dbConn = DatabaseConnectionFactory.createDatabaseConnection('
org.postgresql.Driver','jdbc:postgresql://localhost:5433/dhis','dhis','dhis');
var result=dbConn.executeCachedQuery('select * from t1 limit 10');
//go to the first result
var rets = Packages.java.util.ArrayList();  //定义一个返回列表，保存返回生成的多个源消息，

while(result.next()){

var   res= <person>
                <id/><name/><sex></sex><problemlist><item/></problemlist>
              </person>;
      //get the value from the first column as an integer

      //
      var kk=result.getString('k');
      globalChannelMap.put('pp',kk);
      res['id']=kk;
      res['name']=result.getString('name');

// 也可以这样定义查询。
// var paramList = Packages.java.util.ArrayList();//定义 java List 对象
//paramList.add(kk);
//var res1=dbConn.executeCachedQuery('select * from t2 where k=?',paramList);
    //从关联子表查询明细。
```

```
        var res1=dbConn.executeCachedQuery('select * from t2 where k='+kk);
        //var ss='<problemlist>';
        var cn=0;
        while(res1.next()){
            //ss=ss+'<nn>'+map.get(res1.getString('val'))+'</nn>';
            res['problemlist'].item[cn]=<item>{res1.getString('val')}</item>;
            cn++;
            //res.insertChildAfter(res['problemlist'],<item>{res1.getString('val')}</item>); //  也可以用相关的 xml 节点操作函数。
        }
        //res['problemlist']=new XML(ss+'</problemlist>');
      rets.add(res);
}
        dbConn.close();
// You may access this result below with $('column_name')
return rets;
```

3.11例子：转换多行文本字段到不同的格式

Well I really appreciate the offer for sample **code**, narupley! I would love to show the decision makers a POC of this software. Here is a simplified sample input and output of what I'm trying to do. I don't see a way to post table data here so I apologize if this is a bit hard to parse:

Input:
Header1,Header2,Header3,Header4
Value1,Value2,Value3,Value4
Value1a,Value2a,Value3a,Value4a
Value1b,Value2b,Value3b,Value4b

Output:
Header4,Header5,Header1,Header2,Header3
Value4,,Value1,Value2,Value3
Value4a,,Value1a,Value2a,Value3a
Value4b,,Value1b,Value2b,Value3b

Basically, you set up a File Reader and File Writer (drag over Encoded Data for the **template**), set all your data types to Delimited Text, include a dummy outbound **template** in your destination transformer (like ","), and use **code** like this:

Code:

```
tmp = <delimited/>;

for each (row in msg.row) {
    var newRow = <row/>;
    newRow.column1 = row.column4.toString();
```

```
    newRow.column2 = row.childIndex() > 0 ? '' : 'Header5';
    newRow.column3 = row.column1.toString();
    newRow.column4 = row.column2.toString();
    newRow.column5 = row.column3.toString();
    tmp.appendChild(newRow);
}
```

3.12 定制 java 扩展类并在脚本中调用;

定制的扩展 Java 代码

```
package edu.ut.interfaces;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

public class dataUtils
{
    public String dataMap(String mapFile, String code, String def) throws
FileNotFoundException, IOException
    {
        Properties datmap = new Properties();
        FileInputStream in = new FileInputStream(mapFile);
        datmap.load(in);
        in.close();
        String ootPut = datmap.getProperty(code, def);
        return ootPut;
    }
}
```

打包成 jar 文件并放到 lib/custom. (添加后不要忘记重启医易通服务器).

在 javascript 规则中调用:

Code:

```
//dataMap args:   <look-up file>, <value to look up>, <default value returned if
lookup value is not found>
//The lookup file needs to be formatted as key:value.   You cannot use a comma
//   between the key and value pairs!
var lookup = msg['PID']['PID.5']['PID.5.1'].toString();   // or some key you want to
```

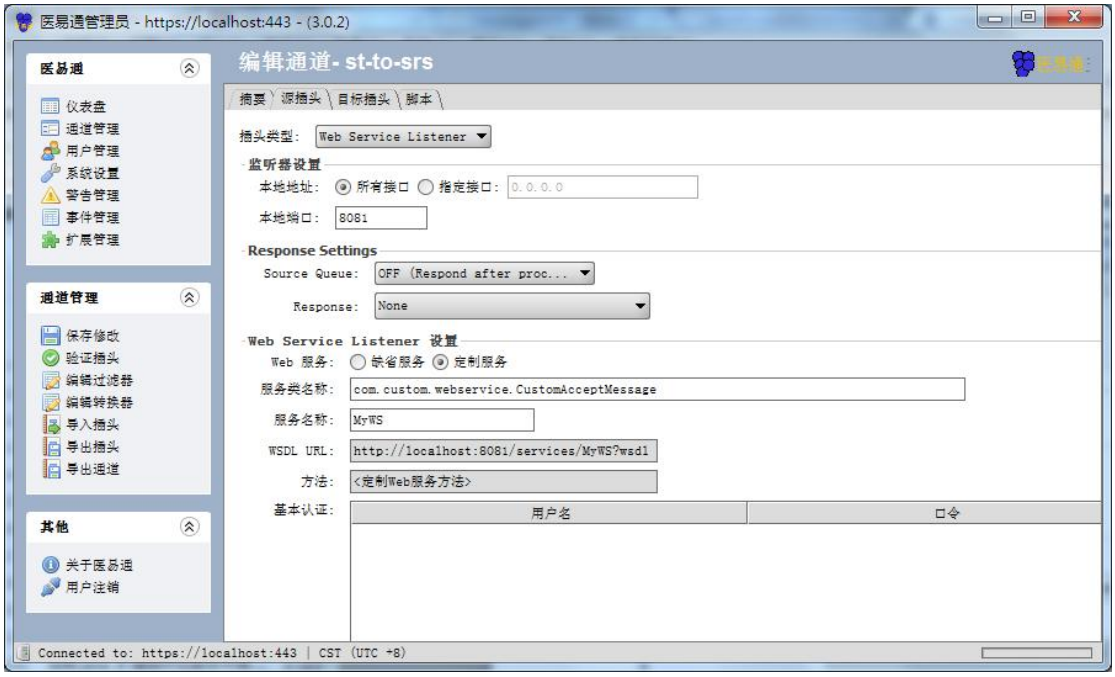
```
lookup in a key:value file
var object = new Packages.edu.ut.interfaces.dataUtils();
var t = object.dataMap("c:/temp/data/misyslocnmap.dat",lookup,"66666");
```

3.13 定制 WebService(3.0.1)

创建一个新的 java 工程，新建一个类，扩展类：
com.woohe.connect.connectors.ws.AcceptMessage 即可。
例子：

```
package com.custom.webservice;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import com.woohe.connect.connectors.ws.AcceptMessage;
import com.woohe.connect.connectors.ws.WebServiceReceiver;

@WebService
public class CustomAcceptMessage extends AcceptMessage {

    public CustomAcceptMessage(WebServiceReceiver webServiceReceiver) {
        super(webServiceReceiver);
    }

    @WebMethod(action = "sample_operation")
    public String operation(@WebParam(name = "param_name") String param)
{
        // implement the web service operation here
        return param;
    }

    @WebMethod(action = "add")
    public int add(@WebParam(name = "i") int i, @WebParam(name
= "j") int j) {
        int k = i + j;
        return k;
    }
}
```

将该工程导出为一个 jar 文件，然后加到你的医易通定制库目录 **custom-lib** ，重启医易通即可。 然后你可以创建你自己定制的 web 服务监控器。



3.14 定制响应

响应可以通过在响应映射里放置响应变量来定义，然后在源插头指定响应来自该响应变量即可。

responseMap.put('key','value')

For 3.0：

```
return ResponseFactory.getSentResponse(message.getConnectorMessages().get(0).getEncodedData());
```

3.15 后处理器例子

本 例子 说明 了如何在 后处理器访问相关 信息 构建 返回。

在医易通3.0中，不是只有源连接器（插头）消息在后处理程序中可用，你可以访问整个全部的消息(可以访问每个连接器的消息)。3.0中提出了一个“合并连接器消息”的概念，基本上它是所有单个连接器的消息组合。大多数字段复制自源连接器消息，但是把响应和通道进行映射结合，通过合并连接器可以获取所有连接器消息 ，你现在可以有两个映射可用(channelMap和responseMap)，现在，可以这样取得合并后的连接器的消息：

代码：

message.getMergedConnectorMessage().get(int metadata)

当然你也可以单独获取某个连接器的消息。getConnectorMessages()函数返回一个包含所有消息的映射。 举个例子,来检索连接器的源编码消息数据:

代码:

message.getConnectorMessages(). get(0).getEncodedData()

后处理器:发送消息后立即执行的脚本。在这里可使用通道的特定变量和全局后处理器。

后处理器例子:

```
// This script executes once after a message has been processed ?
// $('d4') : 通道 id 为 4 的通道返回响应

//channelMap.put('dd7',$('d7'));
//channelMap.put('rr7',$r);
globalChannelMap.put('dd79',message.getConnectorMessages().get(0));

//return message.toString();
//return $('d7'); //返回标识为 7 目标插头响应
return message.getConnectorMessages().get(7).getResponse().getContent();
// 极而言之的例子，获取源消息对应的第 7（id 为 7）的目标插头的响应的内容，格式为 XML 字符串：
// <response>
//   <status>SENT</status>
//   <message>testret</message> // id 为 7 的插头的 返回信息，相当于 $('d7')
//   <statusMessage>JavaScript evaluation successful.</statusMessage>
//</response>
//-------
// 同等于 getResponse(), 有：getStatus（），getRawMessage(),getSent().getContent() (获取发送的消息内容）
// getMessageId(),getEncoded(),getRaw()（原始消息）,getTransformed()(转换器转换后的）
```

4 问题、杂例

5 术语和缩略语定义

a) DNS - Domain Name System. The standard Internet protocol for mapping between textual "domain names" humans use to identify specific computers, and the binary IP addresses computers understand. See http://en.wikipedia.org/wiki/Domain_name_system for more

details.

b) E4X (ECMAScript for XML) - "a set of programming language extensions adding native XML support to ECMAScript [JavaScript]." See http://www.ecma-international.org/publications/standards/Ecma-357.htm for full details.

c) FTP - File Transfer Protocol. An older, common protocol for transferring files between machines over the internet. See http://en.wikipedia.org/wiki/File_Transfer_Protocol for details.

d) HTTP - Hypertext Transfer Protocol. The main protocol underlying the World Wide Web. See http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol for details.

e) IP - Internet Protocol. The core protocol used on the Internet for sending packets of information between machines, on which all other Internet protocols are based. See http://en.wikipedia.org/wiki/Internet_Protocol for details.

f) JavaScript - the primary scripting language for Gateway . Gateway uses the open source Rhino implementation. See http://www.mozilla.org/rhino/ for more information.

g) JDBC - Java Database Connectivity. A Java API for accessing databases. See http://en.wikipedia.org/wiki/JDBC for more information.

h) JMS - Java Message Service. A messaging standard for Java applications. See http://java.sun.com/developer/technicalArticles/Ecommerce/jms/ for an overview.

i) LLP - Lower Level Protocol. The standard protocol used for exchanging HL7 messages over networks. See http://www.hl7.org/ for more information.

j) Mule - an open source messaging platform based on Enterprise Service Bus (ESB) concepts used as the core of Gateway . See http://mule.mulesource.org/display/MULE/Home for more information.

k) ODBC - Open Database Connectivity. A standard language-neutral API for database access. See http://en.wikipedia.org/wiki/Open_database_connectivity for more information.

l) PDF - Portable Document Format. A common file format for formatted text document interchange originally created by Adobe Systems and since recast as an open standard. See http://en.wikipedia.org/wiki/Portable_Document_Format for more information.

m) RTF - Rich Text Format. A proprietary file format for formatted text documented interchange created by Microsoft. See http://en.wikipedia.org/wiki/Rich_text_format for more information.

n) SMTP - Simple Mail Transfer Protocol. The standard Internet protocol for sending email messages. See http://en.wikipedia.org/wiki/Smtp for more information.

o) SOAP - Simple Object Access Protocol. See http://en.wikipedia.org/wiki/SOAP for an overview.

p) TCP - Transmission Control Protocol. The standard connection-based communication protocol on which most Internet protocols are based.

q) URI - Uniform Resource Identifier. A globally unique (across all web services) name for a web-accessible resource of some kind. See http://en.wikipedia.org/wiki/Uniform_Resource_Identifier for the gory technical details.

r) URL - Uniform Resource Locator. A global unique (across all web services) location where a web-accessible resource may be found. See http://en.wikipedia.org/wiki/Uniform_Resource_Locator for the gory technical details.

s) Velocity. An open source template processing engine used by Gateway . See

http://en.wikipedia.org/wiki/Apache_Velocity.

t) WSDL - Web Services Description Language. A standard XML-based language for describing web services. See http://en.wikipedia.org/wiki/Web_Services_Description_Language for technical details.

u) XML - Extensible Markup Language. An open standard for a general-purpose markup language, now commonly used to store various kinds of data in a semi-human-readable way. See http://en.wikipedia.org/wiki/XML for technical details.

参考资料：

1) E4X 教程：
   a) http://wso2.com/project/mashup/0.2/docs/e4xquickstart.html
   b) https://developer.mozilla.org/en-US/docs/Archive/Web/E4X_tutorial
2) 其它资料：